INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

(Approved by AICTE | NAAC Accreditation with 'A' Grade | Accredited by NBA | Affiliated to JNTUH)

Dundigal, Hyderabad - 500 043, Telangana

OUTCOME BASED EDUCATION WITH CHOICE BASED CREDIT SYSTEM

BACHELOR OF TECHNOLOGY

INFORMATION TECHNOLOGY

ACADEMIC REGULATIONS, COURSE CATALOGUE and SYLLABUS BT23

B.Tech Regular Four Year Degree Program
(for the batches admitted from the academic year 2023 - 2024)

&

B.Tech (Lateral Entry Scheme)
(for the batches admitted from the academic year 2024 - 2025)

These rules and regulations may be altered / changed from time to time by the academic council

FAILURE TO READ AND UNDERSTAND THE RULES IS NOT AN EXCUSE

VISION

To bring forth professionally competent and socially sensible engineers, capable of working across cultures meeting the global standards ethically.

MISSION

To provide students with an extensive and exceptional education that prepares them to excel in their profession, guided by dynamic intellectual community and be able to face the technically complex world with creative leadership qualities.

Further, be instrumental in emanating new knowledge through innovative research that emboldens entrepreneurship and economic development for the benefit of wide spread community.

QUALITY POLICY

Our policy is to nurture and build diligent and dedicated community of engineers providing a professional and unprejudiced environment, thus justifying the purpose of teaching and satisfying the stake holders.

A team of well qualified and experienced professionals ensure quality education with its practical application in all areas of the Institute.

PROGRAM OUTCOMES (PO's)

Engineering Graduates will be able to:

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-Long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CONTENTS

"Take up one idea.
Make that one idea your life-think of it, dream of it, live on that idea.
Let the brain muscles, nerves, every part of your body be full of that idea and just leave every other idea alone. This is the way to success"
Swami Vivekananda

PRELIMINARY DEFINITIONS AND NOMENCLATURES

AICTE: Means All India Council for Technical Education, New Delhi.

Autonomous Institute: Means an institute designated as Autonomous by University Grants Commission (UGC), New Delhi in concurrence with affiliating University (Jawaharlal Nehru Technological University, Hyderabad) and State Government.

Academic Autonomy: Means freedom to an institute in all aspects of conducting its academic programs, granted by UGC for Promoting Excellence.

Academic Council: The Academic Council is the highest academic body of the institute and is responsible for the maintenance of standards of instruction, education and examination within the institute. Academic Council is an authority as per UGC regulations and it has the right to take decisions on all academic matters including academic research.

Academic Year: It is the period necessary to complete an actual course of study within a year. It comprises two main semesters i.e., (one odd + one even) and one supplementary semester.

Branch: Means specialization in a program like B.Tech degree program in Aeronautical Engineering, B.Tech degree program in Computer Science and Engineering etc.

Board of Studies (BOS): BOS is an authority as defined in UGC regulations, constituted by Head of the Organization for each of the departments separately. They are responsible for curriculum design and updation in respect of all the programs offered by a department.

Backlog Course: A course is considered to be a backlog course, if the student has obtained a failure grade (F) in that course.

Basic Sciences: The courses offered in the areas of Mathematics, Physics, Chemistry etc., are considered to be foundational in nature.

Betterment: Betterment is a way that contributes towards improvement of the students' grade in any course(s). It can be done by either (a) re-appearing or (b) re-registering for the course.

Commission: Means University Grants Commission (UGC), New Delhi.

Choice Based Credit System: The credit based semester system is one which provides flexibility in designing curriculum and assigning credits based on the course content and hours of teaching along with provision of choice for the student in the course selection.

Certificate Course: It is a course that makes a student to have hands-on expertise and skills required for holistic development in a specific area/field.

Compulsory course: Course required to be undertaken for the award of the degree as per the program.

Continuous Internal Examination: It is an examination conducted towards sessional assessment.

Core: The courses that are essential constituents of each engineering discipline are categorized as professional core courses for that discipline.

Course: A course is offered by a department for learning in a particular semester.

Course Outcomes: The essential skills that need to be acquired by every student through a course.

Credit: A credit is a unit that gives weight to the value, level or time requirements of an academic course. The number of 'Contact Hours' in a week of a particular course determines its credit value. One credit is equivalent to one lecture/tutorial hour per week.

Credit point: It is the product of grade point and number of credits for a course.

Cumulative Grade Point Average (CGPA): It is a measure of cumulative performance of a student over all the completed semesters. The CGPA is the ratio of total credit points secured by a student in various courses in all semesters and the sum of the total credits of all courses in all the semesters. It is expressed up to two decimal places.

Curriculum: Curriculum incorporates the planned interaction of students with instructional content, materials, resources, and processes for evaluating the attainment of Program Educational Objectives.

Department: An academic entity that conducts relevant curricular and co-curricular activities, involving both teaching and non-teaching staff, and other resources in the process of study for a degree.

Detention in a Course: Student who does not obtain minimum prescribed attendance in a course shall be detained in that particular course.

Dropping from Semester: Student who doesn't want to register for any semester can apply in writing in prescribed format before the commencement of that semester.

Elective Course: A course that can be chosen from a set of courses. An elective can be Professional Elective and / or Open Elective.

Evaluation: Evaluation is the process of judging the academic performance of the student in her/his courses. It is done through a combination of continuous internal assessment and semester end examinations.

Experiential Engineering Education (ExEEd): Engineering entrepreneurship requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Our students require sufficient skills to innovate in existing companies or create their own.

Grade: It is an index of the performance of the students in a said course. Grades are indicated by alphabets.

Grade Point: It is a numerical weight allotted to each letter grade on a 10 - point scale.

Honours: An Honours degree typically refers to a higher level of academic achievement at an undergraduate level.

Institute: Means Institute of Aeronautical Engineering, Hyderabad unless indicated otherwise by the context.

Massive Open Online Courses (MOOC): MOOC courses inculcate the habit of self learning. MOOC courses would be additional choices in all the elective group courses.

Minor: Minor are coherent sequences of courses which may be taken in addition to the courses required for the B.Tech degree.

Pre-requisite: A specific course, the knowledge of which is required to complete before student register another course at the next grade level.

Professional Elective: It indicates a course that is discipline centric. An appropriate choice of minimum number of such electives as specified in the program will lead to a degree with specialization.

Program: Means, UG degree program: Bachelor of Technology (B.Tech); PG degree program: Master of Technology (M.Tech) / Master of Business Administration (MBA).

Program Educational Objectives: The broad career, professional and personal goals that every student will achieve through a strategic and sequential action plan.

Project work: It is a design or research-based work to be taken up by a student during his/her final year to achieve a particular aim. It is a credit based course and is to be planned carefully by the student.

Re-Appearing: A student can reappear only in the semester end examination for theory component of a course to the regulations contained herein.

Registration: Process of enrolling into a set of courses in a semester of a program.

Regulations: The regulations, common to all B.Tech programs offered by Institute, are designated as "BT23" and are binding on all the stakeholders.

Semester: It is a period of study consisting of 16 weeks of academic work equivalent to normally minimum of 90 working days. Odd semester commences usually in July and even semester in December of every year.

Semester End Examinations: It is an examination conducted for all courses offered in a semester at the end of the semester.

S/he: Means “she” and “he” both.

Student Outcomes: The essential skill sets that need to be acquired by every student during her/his program of study. These skill sets are in the areas of employability, entrepreneurial, social and behavioral.

University: Means Jawaharlal Nehru Technological University Hyderabad (JNTUH), Hyderabad, is an affiliating University.

Withdraw from a Course: Withdrawing from a course means that a student can drop from a course within the first two weeks of odd or even semester (deadlines are different for summer sessions). However, s/he can choose a substitute course in place of it, by exercising the option within 5 working days from the date of withdrawal.

PREFACE

Dear Students,

The focus at IARE is to deliver value-based education with academically well qualified faculty and infrastructure. It is a matter of pride that IARE continues to be the preferred destination for students to pursue an engineering degree.

In the year 2015, IARE was granted academic autonomy status by University Grants Commission, New Delhi under Jawaharlal Nehru Technology University Hyderabad. From then onwards, our prime focus is on developing and delivering a curriculum which caters to the needs of various stakeholders. The curriculum has unique features enabling students to develop critical thinking, solve problems, analyze socially relevant issues, etc. The academic cycle designed on the basis of Outcome Based Education (OBE) strongly emphasizes continuous improvement and this has made our curriculum responsive to current requirements.

The curriculum at IARE has been developed by experts from academia and industry and it has unique features to enhance problem solving skills apart from academic enrichment. The curriculum of B.Tech program has been thoroughly revised as per AICTE / UGC / JNTUH guidelines and have incorporated unique features such as competency training / coding, industry driven elective, internship and many more. The curriculum is designed in a way so as to impart engineering education in a holistic approach towards Excellence.

I hope you will have a fruitful stay at IARE.

Dr. L V Narasimha Prasad
Principal

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

ACADEMIC REGULATIONS – BT23

B.Tech. Regular Four-Year Degree Program
(for the batches admitted from the academic year 2023 - 2024)
&
B.Tech. (Lateral Entry Scheme)
(for the batches admitted from the academic year 2024 - 2025)

For pursuing four year undergraduate Bachelor of Technology (B.Tech) degree program of study in engineering offered by Institute of Aeronautical Engineering under Autonomous status.

A student shall undergo the prescribed courses as given in the program curriculum to obtain his / her degree in major in which he/she is admitted with 160 credits in the entire program of 4 years. Additional 20/18 credits can be acquired for the degree of B.Tech with **Honours or Minor in Engineering**. These additional 20/18 credits will have to be acquired with massive open online courses (MOOCs) / courses offered by the respective department, to tap the zeal and excitement of learning beyond the classrooms. This creates an excellent opportunity for students to acquire the necessary skill set for employability through massive open online courses where the rare expertise of world-famous experts from academics and industry are available.

Separate certificate will be issued in addition to major degree program mentioning that the student has cleared Honours / Minor specialization in respective courses.

1. CHOICE BASED CREDIT SYSTEM

The credit-based semester system provides flexibility in designing program curriculum and assigning credits based on the course content and hours of teaching. The Choice Based Credit System (CBCS) provides a ‘cafeteria’ type approach in which the students can take courses of their choice, learn at their own pace, undergo additional courses and acquire more than the required credits, and adopt an interdisciplinary approach to learning.

A course defines learning objectives and learning outcomes and comprises lectures / tutorials / laboratory work / field work / project work / seminars / assignments / MOOCs / alternative assessment tools / presentations / self-study etc., or a combination of some of these. Under the CBCS, the requirement for awarding a degree is prescribed in terms of number of credits to be completed by the students.

2. MEDIUM OF INSTRUCTION

The medium of instruction shall be **English** for all courses, examinations, seminar presentations and project work. The program curriculum will comprise courses of study as given in course structure, in accordance with the prescribed syllabi.

3. PROGRAMS OFFERED

Presently, the institute is offering Bachelor of Technology (B.Tech) degree programs in eleven disciplines. The various programs and their two-letter unique codes are given in Table 1.

Table 1: B.Tech Programs offered

S. No	Name of the Program	Title	Code
1	Aeronautical Engineering	AE	07
2	Computer Science and Engineering	CS	05

3	Computer Science and Engineering (AI & ML)	CA	34
4	Computer Science and Engineering (Data Science)	CD	35
5	Computer Science and Engineering (Cyber Security)	CC	36
6	Information Technology	IT	06
7	Electronics and Communication Engineering	EC	04
8	Electrical and Electronics Engineering	EE	02
9	Mechanical Engineering	ME	03
10	Civil Engineering	CE	01

4. SEMESTER STRUCTURE

Each academic year is divided into two semesters, **ODD and EVEN** semester. Both the semesters have regular class work.

4.1 Each semester shall be of 21 weeks (Table 2) duration, and this period includes time for course registration, regular class work, examination preparation, and conduction of examinations.

4.2 Each semester shall have a minimum of 90 Instructional / working days.

4.3 The academic calendar for both Odd and Even semester shown in Table 2 is declared at the beginning of the academic year.

Table 2: Academic Calendar

FIRST SEMESTER (21 weeks)	I Spell Instruction Period	8 weeks	19 weeks
	I Continuous Internal Assessment Examinations (Mid-term)	1 week	
	II Spell Instruction Period	8 weeks	
	II Continuous Internal Assessment Examinations (Mid-term)	1 week	
	Preparation and Practical Examinations	1 week	
	Semester End Examinations		2 weeks
Semester Break and Supplementary Exams			5 weeks
SECOND SEMESTER (21 weeks)	I Spell Instruction Period	8 weeks	19 weeks
	I Continuous Internal Assessment Examinations (Mid-term)	1 week	
	II Spell Instruction Period	8 weeks	
	II Continuous Internal Assessment Examinations (Mid-term)	1 week	
	Preparation and Practical Examinations	1 week	
	Semester End Examinations		2 weeks
Semester Break and Supplementary Exams			5 weeks

4.4 Students admitted on transfer from JNTUH affiliated institutes, Universities and other institutes in the courses in which they are required to earn credits so as to be on par with regular students as prescribed by concerned 'Board of Studies'.

5 REGISTRATION / DROPPING / WITHDRAWAL

The academic calendar includes important academic activities to assist the students and the faculty. These include, dates assigned for registration of courses, dropping of courses and withdrawal from courses. This enables the students to be well prepared and take full advantage of the flexibility provided by the credit system.

5.1. Each student has to compulsorily register for course work at the beginning of each semester as per the schedule mentioned in the Academic Calendar. It is compulsory for the student to register for courses in time. The registration will be organized departmentally under the supervision of the Head of the department.

5.2. In ABSENTIA, registration will not be permitted under any circumstances.

5.3. At the time of registration, students should have cleared all the dues of Institute and Hostel for the previous semesters, paid the prescribed fees for the current semester and not been debarred from the institute for a specified period on disciplinary or any other ground.

5.4. In the first two semesters, the prescribed course load per semester is fixed and is mandated to register all courses. Withdrawal / dropping of courses in the first and second semester is not allowed.

5.5. In all semesters, the average load is 20 credits / semester, with its minimum and maximum limits being set at 16 and 24 credits. This flexibility enables students (**from IV semester onwards**) to cope-up with the course work considering the academic strength and capability of student.

Note: A course may be offered to the students, only if a minimum of 20 students opt for it.

5.6. Dropping of Courses:

Within one week after the last date of first continuous internal examination, the student may in consultation with his / her faculty mentor / adviser, drop one or more courses without prejudice to the minimum number of credits as specified in clause 5.5. The dropped courses are not recorded in the memorandum of grades. Student must complete the dropped course(s) by registering in the forthcoming semester in order to earn the required credits.

5.7. Withdrawal from Courses:

A student is permitted to withdraw from a course before commencement first continuous internal examination. Such withdrawals will be permitted without prejudice to the minimum number of credits as specified in clause 5.5. A student cannot withdraw a course more than once and withdrawal of reregistered courses is not permitted.

6. CREDIT SYSTEM

The B.Tech Program shall consist of a number of courses and each course shall be assigned with credits. The curriculum shall comprise Program Core Courses (PCC), Program Elective Courses (PEC), Open Elective Courses (OEC), Laboratory Courses, Mandatory Courses (MC), Value Added Courses (VAC), Experiential Engineering Education (ExEEd), Internship and Project work.

Depending on the complexity and volume of the course, the number of contact periods per week will be assigned. Each theory and laboratory course carries credits based on the number of hours / week.

- Contact classes (Theory): 1 credit per lecture hour per week, 1 credit per tutorial hour per week.
- Laboratory hours (Practical): 1 credit for 2 practical hours per week.
- Project work: 1 credit for 2 hours of project work per week.
- Experiential Engineering Education (ExEEd): 1 credit for two per hours.
- Mandatory courses / Value added courses : No credit is awarded.

Credit distribution for courses offered is given in Table 3.

Table 3: Credit distribution

S. No	Course	Hours	Credits
1	Theory courses	2 / 3 / 4	2 / 3 / 4
2	Program elective courses / Open elective courses	3 / 2	3 / 2
3	Laboratory courses	2 / 3 / 4	1 / 1.5 / 2
4	Mandatory course / Value added course	-	0
5	Project Work: Phase - I and II	-	14
6	Full Semester Internship (FSI) Project work	-	14

Major benefits of adopting the credit system are listed below:

- Quantification and uniformity in the listing of courses for all programs at institute, like core, electives and project work.
- Ease of allocation of courses under different heads by using their credits to meet national /international practices in technical education.
- Convenience to specify the minimum / maximum limits of course load and its average per semester in the form of credits to be earned by a student.
- Flexibility in program duration for students by enabling them to pace their course load within minimum/maximum limits based on their preparation and capabilities.
- Wider choice of courses available from any department of the same institute or even from other similar institute, either for credit or for audit.
- Improved facility for students to optimize their learning by availing of transfer of credits earned by them from one College to another.

7. CURRICULAR COMPONENTS

Courses in a curriculum may be of three kinds: **Foundation / Skill, Program core courses, Program elective courses and Open elective courses.**

Foundation / Skill Course:

Foundation courses are the courses based upon the content leads to enhancement of skill and knowledge as well as value based and are aimed at man making education. Skill courses are those areas in which one needs to develop a set of skills to learn anything at all. They are fundamental to learning any course.

Program core courses (PCC):

There may be a program core course in every semester. This is the course which is to be compulsorily studied by a student as a core requirement to complete the requirement of a program in the said discipline of study.

Program elective courses (PEC) / Open elective courses (OEC):

Electives provide breadth of experience in respective branch and application areas. The program elective course(s) is a course which can be chosen from a pool of courses. It may be:

- Supportive to the discipline of study
- Providing an expanded scope
- Enabling an exposure to some other discipline / domain
- Nurturing student’s proficiency / skill.

An elective may be program elective, is a discipline centric focusing on those courses which add generic proficiency to the students or may be Open elective course, chosen from unrelated disciplines.

There is list of professional elective courses; students can choose not more than two courses from each track. Overall, students can opt for six professional elective courses which suit their project work in consultation with the faculty advisor / mentor. Nevertheless, one course from each of the three open electives has to be selected. A student may also opt for more elective courses in his/her area of interest.

Every course of the B.Tech program will be placed in one of the eight categories with minimum credits as listed in the Table 4.

Table 4: Category Wise Distribution of Credits

S. No	Category	Breakup of Credits
1	Humanities and Social Sciences (HSMC), including Management.	07
2	Basic Science Courses (BSC) including Mathematics, Physics and Chemistry.	28
3	Engineering Science Courses (ESC), including Workshop, Drawing, ExEEd, Basics of Electrical / Electronics / Mechanical / Computer Engineering.	09

4	Program Core Courses (PCC), relevant to the chosen specialization / branch.	75
5	Program Elective Courses (PEC), relevant to the chosen specialization / branch.	18
6	Open Elective Courses (OEC), from other technical and / or emerging course areas.	09
7	Project work (PROJ) / Full Semester Internship (FSI) Project work	14
8	Mandatory Courses (MC) / Value Added Courses (VAC)	Non-Credit
TOTAL		**160**

Semester wise course break-up

Following are the **TWO** models of course structure out of which any student shall choose or will be allotted with one model based on their academic performance.

i. Full Semester Internship (FSI) Model and
ii. Non Full Semester Internship (NFSI) Model

A student who secures a minimum CGPA of 7.5 upto IV semester with **no current arrears** and maintains the CGPA of 7.5 till VI semester shall be eligible to opt for FSI. Students can opt full semester internship (FSI) either in VII or VIII semesters. In Non-FSI Model, all the selected students shall carry out the course work and project work as specified in the course catalogue.

8. EVALUATION METHODOLOGY

Total marks for each course shall be based on Continuous Internal Assessment (CIA) and Semester End Examinations (SEE). There shall have a uniform pattern of 40:60 for CIA and SEE of both theory and practical courses. The institute shall conduct multiple continuous internal assessments (CIA) for theory courses. All the performances of a student shall be considered for Continuous Internal Assessment (CIA) marks.

Table 5: Outline for Continuous Internal Assessments (CIA-1 and CIA-2) and SEE:

Activities	CIA-1	CIA-2	SEE	Total Marks
Continuous Internal Examination (CIE)	10 marks	10 marks		20 marks
Definitions and Terminology / Quiz	5 marks	5 marks		10 marks
Tech Talk / Assignment	5 marks	5 marks		10 marks
Semester End Examination (SEE)			60 marks	60 marks
Total	--	--	100 marks	

8.1 Continuous Internal Assessments (CIA-1 and CIA-2)

Assessment is an ongoing process that begins with establishing clear and measurable expected outcomes of student learning, provides students with sufficient opportunities to achieve those outcomes, and concludes with gathering and interpreting evidence to determine how well students' learning matches expectations.

The first component (CIA-1) of assessment is for 10 marks, definitions and terminology / quiz carry 05 marks and 05 marks allotted for Tech talk / Assignments. This assessment and score process should be completed after completing of first 50% of syllabus ($2^{1/2}$) of the course/s and within 45 working days of semester program.

The second component (CIA-2) of assessment is for 10 marks, definitions and terminology / quiz carry 05 marks and 05 marks are allotted for Tech talk / Assignments. This assessment and score process should be completed after completing of remaining 50% of syllabus ($2^{1/2}$) of the course/s and within 45 working days of semester program.

In case of a student who has failed to attend the CIA1 or CIA2 on a scheduled date, shall be deemed that the student has dropped the examination. However, in case a student could not take the test on scheduled date due to genuine reasons, may appeal to the HOD / Principal. The HOD / Principal in consultation with the class in-charge shall decide about the genuineness of the case and decide to conduct Make-Up Examination to such candidate on the date fixed by the Examinations Control Office but before commencement of the concerned semester end examinations.

8.2 Definitions and Terminology / Quiz

Definitions and Terminology: The conduction of definitions and terminology is completely online. The faculty should prepare 30 to 35 questions from each and every module by clearly mentioning the answer. The course handling faculty needs to submit detailed document to the respective department.

Mode of conducting examination: Online through e-Examdesk

Quiz: The faculty should prepare 50 multiple choice questions from each module (50 * 5 = 250 questions). Each question will have four choices / options, fill in the blanks etc. The course handling faculty needs to submit detailed document to the respective department.

Mode of conducting examination: Online through e-Examdesk

8.3 Tech Talk / Assignment

Tech Talk: Technical talks cover a wide range of technical concepts and ideas. For conduction of Tech Talk faculty has to submit latest topics from IEEE, CSI, magazines etc.

Assignments: The assignments develop different skills and increase their knowledge base significantly. It provides the evidence for the faculty that the students have achieved the goals. It helps the faculty to evaluate the student's understanding of the course. The output can be judged using sensory perception (observing, reading, tasting etc.). Faculty should prepare 30 to 35 assignment questions from each module and submit the same to respective department.

8.4 Semester End Examination (SEE)

The semester end examinations (SEE), for theory courses, will be conducted for 60 marks. The syllabus for the theory courses is divided into FIVE modules and each module carries equal weightage in terms of marks distribution. The question paper has eight questions; module one and two has one question each, and two questions each from modules three, four and five. There will be no choice for first and second modules. From third module onwards there will be an "either" "or" choice, and the student should answer either of the two questions. Each question carries 12 marks and has two parts (A and B). Part A is a descriptive type question for 6 marks and Part B a critical thinking question / problem solving question for 6 marks. **The duration of semester end examination is 3 hours.**

8.5 Passing Criteria:

To maintain high standards in all aspects of examinations at the institute, the institute shall follow the standards of passing at CIA (CIA-1 and CIA-2) and SEE for each course. However, the student's performance in a course shall be judged by taking into account the results of CIE and SEE individually and also together, as shown below:

a) A minimum of 35% of marks to be secured by cumulating marks in CIA-1 and CIA-2 for appearing for a SEE theory examination.

b) A minimum of 35% of marks to be scored in SEE for passing a theory course.

c) A minimum of 40% of marks in CIA+ SEE for passing a theory course.

8.5 Supplementary examinations

Supplementary examinations for the odd semester shall be conducted with the regular examinations of even semester and vice versa. In case of failure in any course, a student may be permitted to register for the same course when offered.

Advanced supplementary examination will be conducted for VIII semester courses at the end of the program after declaration of results.

8.6 Laboratory Course

Evaluation methodology of laboratory course (CIA)

Each laboratory courses there shall be a CIA during the semester for 40 marks and 60 marks for SEE. The 40 marks for internal evaluation marks are awarded as follows:

1. A write-up on day-to-day experiment in the laboratory (in terms of aim, components / procedure, expected outcome) which shall be evaluated for **10 marks.**
2. **10 marks** for viva-voce (or) tutorial (or) case study (or) application (or) poster presentation of the course concerned.
3. Internal practical examination conducted by the laboratory teacher concerned shall be evaluated for **10 marks.**
4. The remaining **10 marks** are for Laboratory Report/Project and Presentation, which consists of the Design (or) Software / Hardware Model Presentation (or) App Development (or) Prototype Presentation which shall be evaluated after completion of laboratory course and before semester end practical examination.

Evaluation methodology of laboratory course (SEE)

The Semester End Examination shall be conducted by an external examiner and the laboratory handling faculty. The external examiner shall be appointed from the other colleges which will be decided by the Principal.

The Semester End Examination held for 3 hours. Total 60 marks are divided and allocated as shown below:

1. 10 marks for write-up
2. 15 for experiment / program
3. 15 for evaluation of results
4. 10 marks for presentation on another experiment/program in the same laboratory course and
5. 10 marks for viva-voce on concerned laboratory course.

8.7 Mandatory Courses (MC)

These courses are among the compulsory courses will not carry any credits. However, a pass in each such course during the program shall be necessary requirement for the student to qualify for the award of degree. No marks or letter grades shall be allotted for mandatory/non-credit courses. Its result shall be declared as **"Satisfactory" or "Not Satisfactory"** performance.

8.8 Additional Mandatory Courses for lateral entry B.Tech students

In addition to the non-credit mandatory courses for regular B.Tech students, the lateral entry students shall take up the following three non-credit mandatory bridge courses (one in III semester, one in IV semester and one in V semester) as listed in Table 6. The student shall pass the following non-credit mandatory courses for the award of the degree and must clear these bridge courses before advancing to the VII semester of the program.

Table-6: Additional Mandatory Courses for lateral entry students

S. No	Additional mandatory courses for lateral entry students
1	Dip-Mathematics
2	Dip-English Communication Skills
3	Dip-Programming for Problem Solving / Essentials of Problem Solving

8.9 Value Added Courses (VAC)

1. Introduction

Value-Added courses are not part of the curriculum and designed to provide necessary skills to increase the employability quotient and equip the students with essential skills to succeed in life.

Institute offers a wide variety of value-added Courses which shall be conducted after class hours. These courses shall be conducted by experts or in-house staff and help students stand apart from the rest in the job market by adding further value to their resume. These value-added courses will be mostly independent to each type of the fields.

2. Objectives

Objectives of the value-added course are:

- Provide students an understanding of the expectations of industry.
- Improve employability skills of students.
- Bridge the skill gaps and make students industry ready.
- Provide an opportunity to students develop their inter-disciplinary skills.
- Mould students as job providers rather than job seekers.

3. Designing the Courses

- Before designing the syllabus, the feedback from the employers, alumni and industry people will be analyzed and considered to select and design an appropriate course by identifying the gaps and also understand the expectations for current and emerging trends.

- Any new value-added course developed by a department should be placed before the Board of Studies for approval.
- The course offered should not be the same as any course listed in the curriculum of the respective Program / or any other program offered in the institute.
- A unique course code is to be given for each course.

4. Guidelines for conducting value added courses

- Value Added Course is not mandatory to qualify for any program.
- It is a teacher assisted learning course open to all students without any additional fee.
- Classes for VAC will be conducted beyond the regular class work only.
- A student will be permitted to register only one value added course in a semester.

5. Duration and Venue

- The duration of value-added course should not be less than 30 hours.
- The respective Head of the department shall provide class room/s based on the number of students/batches.

6. Procedure for Registration:

The list of value-added courses shall be displayed in the institute website along with the syllabus, objectives and outcomes. A student shall register for a value-added course offered during the semester by submitting the duly filled in registration form. The Head of the department shall segregate the list of students enrolled for the value-added course and submit the details to Dean of academics before the start of course.

7. Attendance

Value added course handling faculty shall be responsible for the maintenance of attendance and assessment who have registered for the course.

- The record shall contain details of the students' attendance, number of classes attended and also Record shall also contain the organisation of lesson plan of the Course Instructor.
- The record shall be submitted to the Head of the department for monitoring the attendance.
- At the end of the semester, the record shall be duly signed by the course coordinator and the Head of the Department and placed in safe custody for any future verification.
- Each student shall have a minimum of 75% attendance in all the courses of the particular semester

failing which he or she will not be permitted to write the semester end examination.

8. Passing Requirement and Grading

- The passing requirement for value added courses shall be 40% of the marks prescribed for the course.
- A candidate who has not secured a minimum of 40% of marks in a course (internal and end-term) shall reappear for the course in the next semester/year.
- The grades obtained in value-added courses will not be included for calculating the CGPA.

9. Course Completion

- Students will get a certificate after they have registered for, written the exam and successfully passed.
- The students who have successfully completed the value-added Course shall be issued with a Certificate duly signed by the Authorized signatories.

Note: Apart from the above, students can also register and get the value-added course completion certificate by registering the courses from SWAYAM, e-PG patashala (NPTEL).

8.10 Experiential Engineering Education (ExEED)

Engineering entrepreneurship requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Students require sufficient skills to innovate in existing companies or create their own.

This course will be evaluated for a total of 100 marks consisting of 40 marks for internal assessment and 60 marks for semester end Examination. Out of 40 marks of internal assessment, students has to submit Innovative Idea in a team of four members in the given format. The semester end examination for 60 marks shall be conducted internally, students has to present the Innovative Idea and it will be evaluated by internal ExEEd faculty with at least one faculty member as examiner from the industry, both nominated by the Principal from the panel of experts recommended by the Dean-CLET.

Center for Outcome Based Teaching and Learning (OBTL) of the institute design and teach ExL power skills courses, to shape the student's future. All the below mentioned Experiential Engineering Education (ExEED) courses are evaluated for one credit each.

- **ExL – Essential of Innovation:** This course creates platform where students experience a hands-on approach to learning about engineering. It focuses on educating the students about diversified platforms for learning the skills, career development, innovations, entrepreneurship etc. Based on the requirements this course is offered in first or second semester.
- **ExL – Prototype / Model Development**: It covers the application of relevant technologies to create interaction prototypes. Students learn about different kinds of prototyping activities involved in designing low-fidelity and high-fidelity prototypes such as POC models, web pages and mobile interfaces etc. This course introduces key concepts, processes and principles of industry driven digital fabrication in a manufacturing environment. Students will undertake small-scale, team-based project work to create fabricated objects that relate to a local industry, organisation or community need or opportunity.

8.11 Project Work / FSI Project Work

This gives students a platform to experience a research driven career in engineering, while developing a device / systems and publishing in reputed SCI / SCOPUS indexed journals and/or filing an **Intellectual Property** (IPR-Patent/Copyright) to aid communities around the world. Students should work individually as per the guidelines issued by head of the department concerned. The benefits to students of this mode of learning include increased engagement, fostering of critical thinking and greater independence.

The topic should be so selected that the students are enabled to complete the work in the stipulated time with the available resources in the respective laboratories. The scope of the work be handling part of the consultancy work, maintenance of the existing equipment, development of new experiment setup or can be a prelude to the main project with a specific outcome.

Project report will be evaluated for 100 marks in total. Assessment will be done for 100 marks out of which, the supervisor / guide will evaluate for 40 marks based on the work and presentation / execution of the work. Subdivision for the remaining 60 marks is based on publication, report, presentation, execution and viva-voce. Evaluation shall be done by a committee comprising the supervisor, Head of the department, and an examiner nominated by the Principal.

8.12 Skill enhancement project

Students must submit the skill enhancement project report of the specified course which are included in the course catalogue. If the student has failed to submit the report or not reached upto the mark, needs to re-register the course in next semesters till completion.

8.13 Project work

The student's project activity is spread over in VII semester and in VIII semesters. A student shall carry out the project work under the supervision of a faculty member or in collaboration with an Industry, R&D organization or another academic institution / University where sufficient facilities exist to carry out the project work.

Project work (Phase - I) starts in VII semester as it takes a vital role in campus hiring process. Students shall select project titles from their respective logins uploaded by the supervisors at the middle of the VI semester. Two reviews are conducted by department review committee (DRC). Student must submit a project report summarizing the work done up to design phase/prototype by the end of VII semester. The semester end examination for project work (Phase-I) is evaluated based on the project report submitted and a viva-voce exam for 60 marks by a committee comprising the head of the department, the project supervisor and an external examiner nominated by the Principal.

Project Work (Phase - II) starts in VIII semester, and it shall be evaluated for 100 marks out of which 40 marks towards continuous internal assessment and 60 marks for semester end examination. Two reviews are to be conducted by DRC on the progress of the project for 40 marks. The semester end examination shall be based on the final report submitted and a viva-voce exam for 60 marks by a committee comprising the head of the department, the project supervisor and an external examiner nominated by the Principal.

A minimum of 50% of maximum marks shall be obtained to earn the corresponding credits.

8.14 Full Semester Internship (FSI)

FSI is a full semester internship program carry 14 credits. The FSI shall be opted in VII semester or in VIII semester. During the FSI, student has to spend one full semester in an identified industry / firm / R&D organization or another academic institution/University where sufficient facilities exist to carry out the project work.

Following are the evaluation guidelines:

- Quizzes: 2 times
- Quiz #1 - About the industry profile, weightage: 5%
- Quiz #2 - Technical-project related, weightage: 5%
- Seminars - 2 times (once in six weeks), weightage: 7.5% + 7.5%
- Viva-voce: 2 times (once in six weeks), weightage: 7.5% + 7.5%
- Project Report, weightage: 15%
- Internship Diary, weightage: 5 %
- Final Presentation, weightage: 40%

FSI shall be open to all the branches with a ceiling of maximum 10% distributed in both semesters. The selection procedure is:

- Choice of the students.
- CGPA (> 7.5) upto IV semester having no credit arrears.
- Competency Mapping / Allotment.

It is recommended that the FSI Project work leads to a research publication in a reputed Journal / Conference or the filing of patent / design with the patent office, or, the start-up initiative with a sustainable and viable business model accepted by the incubation center of the institute together with the formal registration of the startup.

8.15 Field projects / Internship Academic attachment:

The Field Projects (FP) / Internships are mandatory for the students admitted from the academic year 2023 -24 onwards. It is spreaded over from II semester to VI semester.

Field Project: Field project (FP) integrates theory and practice by providing students with an opportunity to work on real-world challenges. It can be used to learn about the functioning and manufacturing procedures of a factory. Besides this, student can also learn about the geographical factors of the region for the specific products /equipment.

Internship is an integral part of the academic curriculum, it is a learning activity in which a student fortifies and deepens his/her theoretical knowledge and skills attained in the classrooms by integrating with practical activities. It offers the students an opportunity to gain hands-on industrial or organizational exposure; to integrate the knowledge and skills acquired through the coursework; interact with professionals and other interns; and to improve their presentation, writing, and communication skills. Internship often acts as a gateway for final placement for many students.

Table 7: Possibility of availing opportunities during semester breaks.

S.No	Schedule	Duration	Type
1	At the end of II semester / Before commencement of III semester	2 Weeks	Field Project
2	At the end of IV semester / Before commencement of V semester	2 Weeks	Internship
3	At the end of VI semester / Before commencement of VII semester	2 Weeks	Internship

Evaluation Methodology of Field Project / Internships:
The evaluation of the field project / field practicum / Internships will be done before commencement of subsequent semester specified in Table: 7. The students have to submit a detailed report of field project / field practicum / Internships through online portal and also carry hard copy of report with geo-tagged photographs. The committee will evaluate by enclosing their comments like **satisfactory or not satisfactory**. If students get not satisfactory results, reports need to be re-submit in the respective department once again for evaluation.

8.16 Plagiarism index for Project report:

All project reports shall go through the plagiarism check and the plagiarism index has to be less than 20%. Project reports with plagiarism more than 20% and less than 60% shall be asked for resubmission within a stipulated period of six months. Project reports with plagiarism more than 60% shall be rejected.

9. ATTENDANCE REQUIREMENTS AND DETENTION POLICY

9.1 A student shall be eligible to appear for the semester end examinations, if the student acquires a minimum of 75% of attendance in aggregate of all the courses (including attendance in mandatory courses like Environmental Science, Constitution of India, and Gender Sensitization etc..) for that semester. **Two periods** of attendance for each theory course shall be considered, if the student appears for the mid-term examination of that course.

9.2 Shortage of attendance in aggregate upto 10% (65% and above, and below 75%) in each semester may be condoned by the college academic committee on genuine and valid grounds, based on the student's representation with supporting evidence.

9.3 A stipulated fee shall be payable for condoning of shortage of attendance.

9.4 Shortage of attendance below 65% in aggregate shall in NO case be condoned.

9.5 Students whose shortage of attendance is not condoned in any semester are not eligible to take their semester end examinations of that semester. They get detained and their registration for that semester shall stand cancelled, including all academic credentials (internal marks etc.) of that semester. They will not be promoted to the next semester. They may seek re-registration for all those courses registered in that semester in which the student is detained, by seeking re-admission into that semester as and when offered; if there are any program electives and / or open electives, the same may also be re- registered if offered. However, if those electives are not offered in later semesters, then alternate electives may be chosen from the **same** set of elective courses offered under that category.

9.6 A student fulfilling the attendance requirement in the present semester shall not be eligible for readmission into the same class.

9.7 A student detained in a semester due to shortage of attendance may be re-admitted in the same semester in the next academic year for fulfillment of academic requirements. The academic regulations under which a student has been re-admitted shall be applicable. Further, no grade allotments or SGPA/ CGPA calculations will be done for the entire semester in which the student has been detained.

9.8 A student detained due to lack of credits, shall be promoted to the next academic year only after acquiring the required number of academic credits. The academic regulations under which the student has been readmitted shall be applicable to him.

10. CONDUCT OF SEMESTER END EXAMINATIONS AND EVALUATION

10.1 Semester end examination shall be conducted by the Controller of Examinations (COE) by inviting question papers from the external examiners.

10.2 COE shall invite 3 - 9 internal / external examiners to evaluate all the semester end examination answer books on a prescribed date(s). Practical laboratory examinations are conducted involving external examiners.

10.3 Examinations control office shall consolidate the marks awarded by examiner/s and award the grades.

11. SCHEME FOR THE AWARD OF GRADE

11.1 A student shall be deemed to have satisfied the minimum academic requirements and earn the credits for each theory course, if s/he secures,

a) Not less than 35% marks for each theory course in the semester end examination, and
b) A minimum of 40% marks for each theory course considering Continuous Internal Assessment (CIA) and Semester End Examination (SEE).

11.2 A student shall be deemed to have satisfied the minimum academic requirements and earn the credits for each Laboratory / Project work / FSI Project work, if s/he secures,

a) Not less than 40% marks for each Laboratory / Project work / FSI project work course in the semester end examination,
b) A minimum of 40% marks for each Laboratory / Project work / FSI project work course considering both internal and semester end examination.

11.3 If a candidate fails to secure a pass in a particular course, it is mandatory that s/he shall register and reappear for the examination in that course during the next semester when examination is conducted in that course. It is mandatory that s/he should continue to register and reappear for the examination till s/he secures a pass.

11.4 A student shall be declared successful or ‘passed’ in a semester, if he secures a Grade Point ≥ 5 (‘C’ grade or above) in every course in that semester (i.e. when the student gets an SGPA ≥5.0 at the end of that particular semester); and he shall be declared successful or ‘passed’ in the entire under graduate programme, only when gets a CGPA ≥5.0 for the award of the degree as required.

12. LETTER GRADES AND GRADE POINTS

12.1 Performances of students in each course are expressed in terms of marks as well as in Letter Grades based on absolute grading system. The UGC recommends a 10-point grading system with the following letter grades as given in the Table 8.

Table-8: Grade Points Scale (Absolute Grading)

% of Marks Secured in a Course (Class Intervals)	Letter Grade	Grade Point
Greater than or equal to 90%	O (Outstanding)	10
80 and less than 90%	A+ (Excellent)	9
70 and less than 80%	A (Very Good)	8
60 and less than 70%	B+ (Good)	7
50 and less than 60%	B (Average)	6
40 and less than 50%	C (Pass)	5
Below 40%	F (Fail)	0
Absent	AB (Absent)	0

12.2 A student is deemed to have passed and acquired to correspondent credits in particular course if s/he obtains any one of the following grades: "O", "A+", "A", "B+", "B", "C".

12.3 A student obtaining Grade F shall be considered Failed and will be required to reappear in the examination.

12.4 For non credit courses, 'PP' or "NP" is indicated instead of the letter grade and this will not be counted for the computation of SGPA / CGPA.

12.5 At the end of each semester, the institute issues grade sheet indicating the SGPA and CGPA of the student. However, grade sheet will not be issued to the student if s/he has any outstanding dues.

Table 09: Percentage Equivalence of Grade Points (for a 10 – Point Scale)

Grade Point	Percentage of Marks / Class
5.5	50
6.0	55
6.5	60
7.0	65
7.5	70
8.0	75

Note:

(1) The following Formula for Conversion of CGPA to percentage of marks to be used only after a student has successfully completed the program:

Percentage of Marks = $(CGPA – 0.5) \times 10$

(2) Class designation:

$\geq$75% (First Class with Distinction),

$\geq$ 60% and <75 % (First Class),

<60 % and $\geq$50% (Second Class),

$\geq$40% and <50% (Pass Class).

(3) The SGPA will be computed and printed on the Memorandum of Grades only if the candidate passes in all the courses offered and gets minimum C grade in all the courses.

(4) CGPA is calculated only when the candidate passes in all the courses offered in all the semesters.

13. COMPUTATION OF SGPA AND CGPA

The UGC recommends to compute the Semester Grade Point Average (SGPA) and Cumulative Grade Point Average (CGPA). The credit points earned by a student are used for calculating the Semester Grade Point Average (SGPA) and the Cumulative Grade Point Average (CGPA), both of which are important performance indices of the student. SGPA is equal to the sum of all the total points earned by the student in a given semester divided by the number of credits registered by the student in that semester. CGPA gives the sum of all the total points earned in all the previous semesters and the current semester divided by the number of credits registered in all these semesters. Thus,

$$SGPA = \sum_{i=1}^{n}(C_i G_i) / \sum_{i=1}^{n} C_i$$

Where, C_i is the number of credits of the i^{th} course and G_i is the grade point scored by the student in the i^{th} course and n represent the number of courses in which a student is registered in the concerned semester.

$$CGPA = \sum_{j=1}^{m}(C_j S_j) / \sum_{j=1}^{m} C_j$$

Where, S_j is the SGPA of the j^{th} semester and C_j is the total number of credits upto the semester and m represent the number of semesters completed in which a student registered upto the semester.

The SGPA and CGPA shall be rounded off to 2 decimal points and reported in the transcripts.

14.0 ILLUSTRATION OF COMPUTATION OF SGPA AND CGPA

14.1 Illustration for SGPA

Course Name	Course Credits	Grade letter	Grade point	Credit Point (Credit x Grade)
Course 1	4	A	8	4 x 8 = 32
Course 2	4	O	10	4 x 10 = 40
Course 3	4	C	5	4 x 5 = 20
Course 4	3	B	6	3 x 6 = 18
Course 5	3	A+	9	3 x 9 = 27
Course 6	3	C	5	3 x 5 = 15
	21			**152**

Thus, SGPA = 152 / 21= 7.24

14.2 Illustration for calculation of CGPA upto 3rd semester

Semester	Course Title	Credits Allotted	Letter Grade Secured	Corresponding Grade Point (GP)	Credit Points (CP)
I	Course 1	3	A	8	24
I	Course 2	3	O	10	30
I	Course 3	3	B	6	18
I	Course 4	4	A	8	32
I	Course 5	3	A+	9	27
I	Course 6	4	C	5	20
II	Course 7	4	B	6	24
II	Course 8	4	A	8	32
II	Course 9	3	C	5	15
II	Course 10	3	O	10	30
II	Course 11	3	B+	7	21
II	Course 12	4	B	6	24
II	Course 13	4	A	8	32

II	Course 14	3	O	10	30
III	Course 15	2	A	8	16
III	Course 16	1	C	5	5
III	Course 17	4	O	10	40
III	Course 18	3	B+	7	21
III	Course 19	4	B	6	24
III	Course 20	4	A	8	32
III	Course 21	3	B+	7	21
	Total Credits	**69**		**Total Credits**	**518**

Thus, CGPA = 518 / 69= 7.51

- The calculation process of CGPA illustrated above will be followed for each subsequent semester until 8th semester. The CGPA obtained at the end of 8th semester will become the final CGPA secured for entire B.Tech. program.
- For merit ranking or comparison purposes or any other listing, only the rounded off' values of the CGPAs will be used.
- SGPA and CGPA of a semester will be mentioned in the semester Memorandum of Grades if all courses of that semester are passed in first attempt, otherwise the SGPA and CGPA shall be mentioned only on the Memorandum of Grades in which sitting s/he passed his last exam in that semester. However, mandatory courses will not be taken into consideration.

15. REVIEW OF SEE THEORY ANSWER BOOKS

If the examinee is not satisfied with the marks awarded, s/he may apply for revaluation of answer book in prescribed format online within three (3) working days from the date of declaration of result of the examination or issue of the statement of marks, whichever is earlier. The revaluation facility shall be for theory papers only. The revaluation of answer book shall not be permitted in respect of the marks awarded to the scripts of practical examination / project work (including theory part) and in viva voce / oral / comprehensive examinations.

The re-evaluation will be done by a second independent examiner. The result after re-evaluation shall be as follows:

1. The revaluation marks are considered only if the difference between the original award and award on reevaluation is more than equal to 15% of 60 marks (09 marks).
2. If the difference between the original award and the award on reevaluation is more than 20% (12 marks), a third evaluator is to be appointed and the average of two nearest awards (in the range of 15%) shall be considered.

16. PROMOTION POLICIES

The following academic requirements have to be satisfied in addition to the attendance requirements mentioned in item no. 9.

16.1 For students admitted into B.Tech (Regular) program

16.1.1 A student will not be promoted from II semester to III semester unless s/he fulfills the academic requirement of securing 50% of the total credits (rounded to the next lowest integer) from I and II semester examinations, whether the candidate takes the examination(s) or not.

16.1.2 A student will not be promoted from IV semester to V semester unless s/he fulfills the academic requirement of securing 60% of the total credits (rounded to the next lowest integer) upto III semester **or** 60% of the total credits (rounded to the next lowest integer) up to IV semester, from all the examinations, whether the candidate takes the examination(s) or not.

16.1.3 A student shall be promoted from VI semester to VII semester only if s/he fulfills the academic requirements of securing 60% of the total credits (rounded to the next lowest integer) up to V semester **or** 60% of the total credits (rounded to the next lowest integer) up to VI semester from all the examinations, whether the candidate takes the examination(s) or not.

16.1.4 A student shall register for all the 160 credits and earn all the 160 credits. Marks obtained in all the 160 credits shall be considered for the award of the Grade.

16.2 For students admitted into B.Tech (Lateral entry students)

16.2.1 A student will not be promoted from IV semester to V semester unless s/he fulfills the academic requirement of securing 60% of the total credits (rounded to the next lowest integer) up to IV semester, from all the examinations, whether the candidate takes the examination(s) or not.

16.2.2 A student shall be promoted from VI semester to VII semester only if s/he fulfills the academic requirements of securing 60% of the total credits (rounded to the next lowest integer) up to V semester **or** 60% of the total credits (rounded to the next lowest integer) up to VI semester from all the examinations, whether the candidate takes the examination(s) or not.

16.2.3 A student shall register for all the 120 credits and earn all the 120 credits. Marks obtained in all the 120 credits shall be considered for the award of the Grade.

17. GRADUATION REQUIREMENTS

The following academic requirements shall be met for the award of the B.Tech degree.

17.1 Student shall register and acquire minimum attendance in all courses and secure 160 credits (with minimum CGPA of 5.0), for regular program and 120 credits (with minimum CGPA of 5.0), for lateral entry program. **There is NO exemption of credits in any case.**

17.2 A student of a regular program, who fails to earn 160 credits within **eight consecutive academic years** from the year of his/her admission with a minimum CGPA of 5.0, shall forfeit his/her degree and his/her admission stands cancelled.

17.3 A student of a lateral entry program who fails to earn 120 credits within **six consecutive academic years** from the year of his/her admission with a minimum CGPA of 5.0, shall forfeit his/her degree and his/her admission stands cancelled.

18. AWARD OF DEGREE

18.1 Classification of degree will be as follows:

CGPA > 8.0	CGPA ≥ 7.0 and < 8.0	CGPA ≥ 6.0 and < 7.0	CGPA ≥ 5.0 and < 6.0	CGPA < 5.0
First Class with Distinction	First Class	Second Class	Pass Class	Fail

18.2 A student with final CGPA (at the end of the under graduate programme) >8.0, and fulfilling the following conditions - shall be placed in '**first class with distinction**'. However,

a. Should have passed all the courses in '**first appearance**' within the first 4 academic years (or 8 sequential semesters) from the date of commencement of first semester.

b. Should have secured a CGPA >8.0, at the end of each of the 8 sequential semesters, starting from first semester onwards.

c. Should not have been detained or prevented from writing the semester end examinations in any semester due to shortage of attendance or any other reason.

d. A student not fulfilling any of the above conditions with final CGPA >8.0 shall be placed in **'first class'.**

18.3 Students with final CGPA (at the end of the B.Tech program) ≥7.0 but <8.0 shall be placed in **'first class'.**

18.4 Students with final CGPA (at the end of the B.Tech program) ≥6.0 but <7.0, shall be placed in '**second class**'.

18.5 All other students who qualify for the award of the degree (as per item 18), with final CGPA (at the end of the B.Tech program) ≥5.0 but <6.0, shall be placed in '**pass class**'.

18.6 A student with final CGPA (at the end of the B.Tech program) <5.0 will not be eligible for the award of the degree.

18.7 Students fulfilling the conditions listed under item 18.2 alone will be eligible for award of '**Gold Medal**'.

All the candidates who register for the semester end examination will be issued a memorandum of grades sheet by the institute. Apart from the semester wise memorandum of grades sheet, the institute will issue the provisional certificate and consolidated grades memorandum course to the fulfillment of all the academic requirements.

19. B.TECH WITH HONOURS OR MINOR IN ENGINEERING

Students acquiring 160 credits are eligible to get B.Tech degree in Engineering. A student will be eligible to get B.Tech degree with Honours or Minor in Engineering, if s/he completes an additional 20/18 credits (3/4 credits per course). These could be acquired through MOOCs from SWAYAM / NPTEL only. The list for MOOCs will be a dynamic one, as new courses are added from time to time. Few essential skill sets required for employability are also identified year wise. Students interested in doing MOOC courses shall register the course title at their department office at the start of the semester against the courses that are announced by the department. Any expense incurred for the MOOC course / summer program should be met by the students.

Students having no credit arrears and a CGPA of 7.5 or above at the end of the fourth semester are eligible to register for B.Tech (Honours / Minor). After registering for the B.Tech (Honours / Minor) program, if a student fails in any course, s/he will not be eligible for B.Tech (Honours / Minor).

Every department should develop and submit a Honours / Minor – courses list of 5 - 6 theory courses, laboratory and project work.

Honours Certificate for Vertical in his/her OWN Branch for Research orientation; Minor in any other branch for Improving Employability.

Honours will be reflected in the degree certificate as "B.Tech (Honours) in XYZ Engineering". Similarly, Minor as "B.Tech in XYZ Engineering with Minor in ABC".

19.1. B.Tech with Honours

The key objectives of offering B. Tech. with Honors program are:

- To expand the domain knowledge of the students laterally and vertically.
- To increase the employability of undergraduate students with expanded knowledge in one of the core engineering disciplines.
- To provide an opportunity to students to pursue their higher studies in wider range of specializations.

Academic Regulations for B. Tech. Honours degree

1. The weekly instruction hours, internal and external evaluation and award of grades are on par with regular 4-Years B. Tech. program.
2. For B. Tech with Honors program, a student needs to earn additional 20 credits (over and above the required 160 credits for B. Tech degree). All these 20 credits required to be attained for B.Tech Honors degree credits are distributed from V semester to VII semester.
3. After registering for the Honors program, if a student is unable to pass all courses in first attempt and earn the required 20 credits, he/she shall not be awarded Honours degree. However, if the student earns all the required 160 credits of B. Tech., he/she will be awarded only B. Tech degree in the concerned branch.
4. There is no transfer of credits from courses of Honors program to regular B. Tech. degree course & vice versa.
5. These **20 credits** are to be earned from the additional courses offered by the host department in the institute or from closely related departments in the institute as well as from the MOOCS platform (NPTEL only).
6. The choice to opt/take the Honors program is purely on the choice of the students.
7. The student shall be given a choice of withdrawing all the courses registered and/or the credits earned for Honours program at any time; and in that case the student will be awarded only B.Tech. degree in the concerned branch on earning the required credits of 160.

8. The students of every branch can choose Honours program in their respective branches if they are eligible for the Honors program. A student who chooses an **Honors program is not eligible to choose a Minor program and vice-versa.**
9. A student can graduate with Honors if he/she fulfils the requirements for his/her regular B.Tech. program as well as fulfills the requirements for Honours program.
10. The institute shall maintain a record of students registered and pursuing their Honors programs branch-wise.
11. The department shall prepare the time-tables for each Honors program offered at their respective departments without any overlap/clash with other courses of study in the respective semesters.

Eligibility conditions of the students for the B.Tech Honors degree

a) A student can opt for B.Tech. degree with Honors, if she/he passed all courses in first attempt in all the semesters till the results announced and maintaining **7.5 or more CGPA.**

b) If a student fails in any registered course of either B.Tech or Honours in any semester of four years program, he/she will not be eligible for obtaining Honors degree. He will be eligible for only

c) Prior approval of mentor and Head of the Department for the enrolment into Honors program, before commencement of V Semester, is mandatory.

d) If more than **30% of the students** in a branch fulfill the eligibility criteria (as stated above), the number of students given eligibility should be limited to 30%. The criteria to be followed for choosing 30% candidates in a branch may be the CGPA secured by the students till **III** semester.

e) Successful completion of **20 credits** earmarked for Honours program with at least 7.5 CGPA along with successful completion of 160 credits earmarked for regular B. Tech. Program with at least 7.5 CGPA and passing all courses in first attempt gives the eligibility for the award of B.Tech (Honors) degree.

f) For CGPA calculation of B.Tech. course, the **20 credits** of Honours program will not be considered.

Following are the details of such Honors which include some of the most interesting areas in the profession today:

S. No	Department	Honors scheme
1	Aeronautical Engineering	Aerospace Engineering / Space Science etc.
2	Computer Science and Engineering / Information Technology	Big data and Analytics / Cyber Physical Systems, Information Security / Cognitive Science / Artificial Intelligence/ Machine Learning / Data Science / Internet of Things (IoT) / Cyber Security etc.
3	Electronics and Communication Engineering	Digital Communication / Signal Processing / Communication Networks / VLSI Design / Embedded Systems etc.
4	Electrical and Electronics Engineering	Renewable Energy systems / Energy and Sustainability / IoT Applications in Green Energy Systems etc.
5	Mechanical Engineering	Industrial Automation and Robotics / Manufacturing Sciences and Computation Techniques etc.
6	Civil Engineering	Structural Engineering / Environmental Engineering etc.

19.2 B.Tech with Minor in Engineering

The key objectives of offering B.Tech with Minor program are:

- To expand the domain knowledge of the students in one of the other branches of engineering.
- To increase the employability of undergraduate students keeping in view of better opportunity in interdisciplinary areas of engineering & technology.
- To provide an opportunity to students to pursue their higher studies in the inter-disciplinary areas in addition to their own branch of study.

- To offer the knowledge in the areas which are identified as emerging technologies/thrust areas of Engineering.

Advantages of Minor in Engineering

The minors mentioned above are having lots of advantages and a few are listed below:

- To enable students to pursue allied academic interest in contemporary areas.
- To provide an academic mechanism for fulfilling multidisciplinary demands of industries.
- To provide effective yet flexible options for students to achieve basic to intermediate level competence in the Minor area.
- Provides an opportunity to students to become entrepreneurs and leaders by taking business/management minor.
- Combination in the diverse fields of engineering e.g., CSE (Major) + Electronics (Minor) combination increases placement prospects in chip designing companies.
- Provides an opportunity for applicants to pursue higher studies in an inter-disciplinary field of study.
- To increase the overall scope of the undergraduate degrees.

Academic Regulations for B.Tech Degree with Minor programs

1. The weekly instruction hours, internal & external evaluation and award of grades are on par with regular 4-Years B. Tech. program.
2. For B. Tech. with Minor, a student needs to earn additional 18 credits (over and above the required 160 credits for B. Tech degree). The courses are offered from V semester to VII semester only, to obtain minor degree studens required to obtain 18 credits.
3. After registering for the Minor program, if a student is unable to earn all the required 18 credits in a specified duration (twice the duration of the course), he/she shall not be awarded Minor degree. However, if the student earns all the required 160 credits of B.Tech, he/she will be awarded only B. Tech degree in the concerned branch.
4. There is no transfer of credits from Minor program courses to regular B. Tech. degree course & vice versa.
5. These 18 credits are to be earned from the additional courses offered by the host department in the institute as well as from the MOOCs platform.
6. For the course selected under MOOCs platform (NPTEL) following guidelines may be followed:
    a) Prior to registration of MOOCs courses, formal approval of the courses, by the institute is essential, before the issue of approval considers the parameters like the institute / agency which is offering the course, syllabus, credits, duration of the program and mode of evaluation etc.
    b) Minimum credits for MOOCs course must be equal to or more than the credits specified in the Minor course structure provided by the institute.
    c) Only Pass-grade / marks or above shall be considered for inclusion of grades in minor grade memo.
    d) Any expenses incurred for the MOOCs courses are to be met by the students only.
7. The choice to opt / take a Minor program is purely on the choice of the students.
8. The student shall be given a choice of withdrawing all the courses registered and / or the credits earned for Minor program at any time; and in that case the student will be awarded only B. Tech. degree in the concerned branch on earning the required credits of 160.
9. The student can choose only one Minor program along with his / her basic engineering degree. A student who chooses an **Honors program is not eligible to choose a Minor program and vice-versa.**
10. The institute shall maintain a record of students registered and pursuing their Minor programs, minor program-wise and parent branch-wise.

11. The institute / department shall prepare the time-tables for each Minor course offered at their respective institutes without any overlap/clash with other courses of study in the respective semesters.

Eligibility conditions for the student to register for Minor course

a) A student can opt for B.Tech. degree with Minor program if she/he has no active backlogs till III semester at the time of entering into III year I semester.

b) Prior approval of mentor and Head of the Department for the enrolment into Minor program, before commencement of III year I Semester (V Semester), is mandatory.

c) If more than 50% of the students in a branch fulfill the eligibility criteria (as stated above), the number of students given eligibility should be limited to 50%.

20.0 TEMPORARY BREAK OF STUDY FROM THE PROGRAM

20.1 A candidate is normally not permitted to take a break from the study. However, if a candidate intends to temporarily discontinue the program in the middle for valid reasons (such as accident or hospitalization due to prolonged ill health) and to rejoin the program in a later respective semester, s/he shall seek the approval from the Principal in advance. Such application shall be submitted before the last date for payment of examination fee of the semester and forwarded through the Head of the Department stating the reasons for such withdrawal together with supporting documents and endorsement of his / her parent / guardian.

20.2 The institute shall examine such an application and if it finds the case to be genuine, it may permit the student to temporarily withdraw from the program. Such permission is accorded only to those who do not have any outstanding dues / demand at the College / University level including tuition fees, any other fees, library materials etc.

20.3 The candidate has to rejoin the program after the break from the commencement of the respective semester as and when it is offered.

20.4 The total period for completion of the program reckoned from the commencement of the semester to which the candidate was first admitted shall not exceed the maximum period specified in clause 17. The maximum period includes the break period.

20.5 If any candidate is detained for any reason, the period of detention shall not be considered as ‘Break of Study’.

21. TERMINATION FROM THE PROGRAM

The admission of a student to the program may be terminated and the student is asked to leave the institute in the following circumstances:

a. The student fails to satisfy the requirements of the program within the maximum period stipulated for that program.
b. A student shall not be permitted to study any semester more than three times during the entire program of study.
c. The student fails to satisfy the norms of discipline specified by the institute from time to time.

22. WITH-HOLDING OF RESULTS

If the candidate has not paid any dues to the institute / if any case of indiscipline / malpractice is pending against him, the results and the degree of the candidate will be withheld.

23. GRADUATION DAY

The institute shall have its own annual Graduation Day for the award of degrees to the students completing the prescribed academic requirements in each case, in consultation with the University and by following the provisions in the Statute. The college shall institute prizes and medals to meritorious students and award them annually at the Graduation Day. This will greatly encourage the students to strive for excellence in their academic work.

24. DISCIPLINE

Every student is required to observe discipline and decorum both inside and outside the institute and are expected not to indulge in any activity which will tend to bring down the honour of the institute. If a student indulges in malpractice in any of the theory / practical examination, continuous assessment examinations, he/she shall be liable for punitive action as prescribed by the institute from time to time.

25. GRIEVANCE REDRESSAL COMMITTEE

The institute shall form a Grievance Redressal Committee for each course in each department with the Course Teacher and the HOD as the members. The committee shall solve all grievances related to the course under consideration.

26. TRANSITORY REGULATIONS

A candidate, who is detained or has discontinued a semester, on readmission shall be required to do all the courses in the curriculum prescribed for the batch of students in which the student joins subsequently. However, exemption will be given to those candidates who have already passed such courses in the earlier semester(s) he was originally admitted into and substitute courses were offered in place of them as decided by the Board of Studies. However, the decision of the Board of Studies will be final.

a) Transfer candidates (from non-autonomous college affiliated to JNTUH):

A student who is following JNTUH curriculum, transferred from other college to this institute in third semester or subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute courses are offered in their place as decided by the Board of Studies. The student has to clear all his backlog courses up to previous semester by appearing for the supplementary examinations conducted by JNTUH for the award of degree. The total number of credits to be secured for the award of the degree will be the sum of the credits up to the previous semester under JNTUH regulations and the credits prescribed for the semester in which a candidate joined after transfer and subsequent semesters under the autonomous status. The class will be awarded based on the academic performance of a student in the autonomous pattern.

b) Transfer candidates (from an autonomous college affiliated to JNTUH):

A student who has secured the required credits up to previous semester as per the regulations of other autonomous institutions shall also be permitted to be transferred to this institute. A student who is transferred from the other autonomous colleges to this institute in third semester or subsequent semesters shall join with the autonomous batch in the appropriate semester. Such candidates shall be required to pass in all the courses in the program prescribed by the Board of Studies concerned for that batch of students from that semester onwards to be eligible for the award of degree. However, exemption will be given in the courses of the semester(s) of the batch which he had passed earlier and substitute courses are offered in their place as decided by the Board of Studies. The total number of credits to be secured for the award of the degree will be the sum of the credits up to previous semester as per the regulations of the college from which he is transferred and the credits prescribed for the semester in which a candidate joined after transfer and subsequent semesters under the autonomous status. The class will be awarded based on the academic performance of a student in the autonomous pattern.

c) Readmission from R18 / UG20 to BT23 regulations

A student took admission in BT23 Regulations, detained due to lack of required number of credits or shortage of attendance at the end of any semester is permitted to take re-admission at appropriate level under any regulations prevailing in the institute course to the following rules and regulations.

1. Student shall pass all the courses in the earlier scheme of regulations. However, in case of having backlog courses, they shall be cleared by appearing for supplementary examinations conducted under earlier regulations from time to time.

2. After readmission, the student is required to study the courses as prescribed in the new regulations for the re-admitted program at that level and thereafter.
3. If the student has already passed any course(s) of readmitted program in the earlier regulation / semester of study, such courses are exempted.
4. The courses that are not done in the earlier regulations / semester as compared need to be cleared after readmission by appearing for the examinations conducted time to time under the new regulations.
5. In general, after transition, course composition and number of credits / semester shall be balanced between old and new regulations on case to case basis.
6. In case, the students who do not have option of acquiring required credits with the existing courses offered as per the new curriculum under autonomy, credit balance can be achieved by clearing the additional courses offered. The additional courses that are offered can be of theory or laboratory courses.

27. REVISION OF REGULATIONS AND CURRICULUM

The Institute from time to time may revise, amend or change the regulations, scheme of examinations and syllabi if found necessary and on approval by the Academic Council and the Governing Body shall be binding on the students, faculty, staff, all authorities of the Institute and others concerned.

FREQUENTLY ASKED QUESTIONS AND ANSWERS ABOUT AUTONOMY

1. **Who grants Autonomy? UGC, Govt., AICTE or University**
   In case of Colleges affiliated to a university and where statutes for grant of autonomy are ready, it is the respective University that finally grants autonomy but only after concurrence from the respective state Government as well as UGC. The State Government has its own powers to grant autonomy directly to Govt. and Govt. aided Colleges.

2 **Shall IARE award its own Degrees?**
No. Degree will be awarded by Jawaharlal Nehru Technological University, Hyderabad with a mention of the name IARE on the Degree Certificate.

3 **What is the difference between a Deemed University and an Autonomy College?**
A Deemed University is fully autonomous to the extent of awarding its own Degree. A Deemed University is usually a Non-Affiliating version of a University and has similar responsibilities like any University. An Autonomous College enjoys Academic Autonomy alone. The University to which an autonomous college is affiliated will have checks on the performance of the autonomous college.

4 **How will the Foreign Universities or other stake – holders know that we are an Autonomous College?**
Autonomous status, once declared, shall be accepted by all the stake holders. The Govt. of Telangana mentions autonomous status during the First Year admission procedure. Foreign Universities and Indian Industries will know our status through our website.

5 **What is the change of Status for Students and Teachers if we become Autonomous?**
An autonomous college carries a prestigious image. Autonomy is actually earned out of our continued past efforts on academic performances, our capability of self- governance and the kind of quality education we offer.

6 **Who will check whether the academic standard is maintained / improved after Autonomy? How will it be checked?**
There is a built in mechanism in the autonomous working for this purpose. An Internal Committee called Academic Program Evaluation Committee, which will keep a watch on the academics and keep its reports and recommendations every year. In addition the highest academic council also supervises the academic matters. The standards of our question papers, the regularity of academic calendar, attendance of students, speed and transparency of result declaration and such other parameters are involved in this process.

7 **Will the students of IARE as an Autonomous College qualify for University Medals and Prizes for academic excellence?**
No. IARE has instituted its own awards, medals, etc. for the academic performance of the students. However for all other events like sports, cultural on co-curricular organized by the University the students shall qualify.

8 **Can IARE have its own Convocation?**
No. Since the University awards the degree the convocation will be that of the University, but there will be Graduation Day at IARE.

9 **Can IARE give a provisional degree certificate?**
Since the examinations are conducted by IARE and the results are also declared by IARE, the college sends a list of successful candidates with their final Grades and Grade Point Averages including CGPA to the affiliating university. Therefore with the prior permission of the university the institute will be entitled to give the provisional certificate.

10 **Will Academic Autonomy make a positive impact on the Placements or Employability?**
Certainly. The number of students qualifying for placement interviews is expected to improve, due to

rigorous and repetitive classroom teaching and continuous assessment. Also the autonomous status is more responsive to the needs of the industry. As a result therefore, there will be a lot of scope for industry oriented skill development built-in into the system. The graduates from an autonomous college will therefore represent better employability.

11 What is the proportion of Internal and External Assessment as an Autonomous College?

Presently, it is 60% external and 40% internal. As the autonomy matures the internal assessment component shall be increased at the cost of external assessment.

12 Is it possible to have complete Internal Assessment for Theory or Practicals?

Yes indeed. We define our own system. We have the freedom to keep the proportion of external and internal assessment component to choose.

13 Why Credit based Grade System?

The credit based grade system is an accepted standard of academic performance the world over in all Universities. The acceptability of our graduates in the world market shall improve.

14 What exactly is a Credit based Grade System?

The credit based grade system defines a much better statistical way of judging the academic performance. One Lecture Hour per week of Teaching Learning process is assigned One Credit. One hour of laboratory work is assigned half credit. Letter Grades like A, B,C,D, etc. are assigned for a Range of Marks. (e.g. 91% and above is A+, 80 to 90 % could be A etc.) in Absolute Grading System while grades are awarded by statistical analysis in relative grading system. We thus dispense with sharp numerical boundaries. Secondly, the grades are associated with defined Grade Points in the scale of 1 to 10. Weighted Average of Grade Points is also defined Grade Points are weighted by Credits and averaged over total credits in a Semester. This process is repeated for all Semesters and a CGPA defines the Final Academic Performance

15 What are the norms for the number of Credits per Semester and total number of Credits for UG/PG program?

These norms are usually defined by UGC or AICTE. Usually around 25 Credits per semester is the accepted norm.

16 What is a Semester Grade Point Average (SGPA)?

The performance of a student in a semester is indicated by a number called SGPA. The SGPA is the weighted average of the grade points obtained in all the courses registered by the student during the semester.

$$SGPA = \sum_{i=1}^{n} (C_i G_i) / \sum_{i=1}^{n} C_i$$

Where, C_i is the number of credits of the i^{th} course and G_i is the grade point scored by the student in the i^{th} course and i represent the number of courses in which a student registered in the concerned semester. SGPA is rounded to two decimal places.

17 What is a Cumulative Grade Point Average (CGPA)?

An up-to-date assessment of overall performance of a student from the time of his first registration is obtained by calculating a number called CGPA, which is weighted average of the grade points obtained in all the courses registered by the students since he entered the Institute.

$$CGPA = \sum_{j=1}^{m} (C_j S_j) / \sum_{j=1}^{m} C_j$$

Where, S_j is the SGPA of the j^{th} semester and C_j is the total number of credits upto the semester and m represent the number of semesters completed in which a student registered upto the semester. CGPA is rounded to two decimal places.

18 Is there any Software available for calculating Grade point averages and converting the same into Grades?

Yes, The institute has its own MIS software for calculation of SGPA, CGPA, etc.

19 Will the teacher be required to do the job of calculating SGPAs etc. and convert the same into Grades?

No. The teacher has to give marks obtained out of whatever maximum marks as it is. Rest is all done by the computer.

20 Will there be any Revaluation or Re-Examination System?

No. There will double valuation of answer scripts. There will be a makeup Examination after a reasonable preparation time after the End Semester Examination for specific cases mentioned in the Rules and Regulations. In addition to this, there shall be a 'summer term' (compressed term) followed by the End Semester Exam, to save the precious time of students.

21 How fast Syllabi can be and should be changed?

Autonomy allows us the freedom to change the syllabi as often as we need.

22 Will the Degree be awarded on the basis of only final year performance?

No. The CGPA will reflect the average performance of all the semester taken together.

23 What are Statutory Academic Bodies?

Governing Body, Academic Council, Examination Committee and Board of Studies are the different statutory bodies. The participation of external members in everybody is compulsory. The institute has nominated professors from IIT, NIT, University (the officers of the rank of Pro-vice Chancellor, Deans and Controller of Examinations) and also the reputed industrialist and industry experts on these bodies.

24 Who takes Decisions on Academic matters?

The Governing Body of institute is the top academic body and is responsible for all the academic decisions. Many decisions are also taken at the lower level like Boards of Studies. Decisions taken at the Boared of Studies level are to be ratified at the Academic Council and Governing Body.

25 What is the role of Examination committee?

The Examinations Committee is responsible for the smooth conduct of internal, End Semester and make up Examinations. All matters involving the conduct of examinations spot valuations, tabulations preparation of Grade Sheet etc fall within the duties of the Examination Committee.

26 Is there any mechanism for Grievance Redressal?

The institute has grievance redressal committee, headed by Dean - Student affairs and Dean - IQAC.

27 How many attempts are permitted for obtaining a Degree?

All such matters are defined in Rules & Regulation.

28 Who declares the result?

The result declaration process is also defined. After tabulation work wherein the SGPA, CGPA and final Grades are ready, the entire result is reviewed by the Moderation Committee. Any unusual deviations or gross level discrepancies are deliberated and removed. The entire result is discussed in the Examinations and Result Committee for its approval. The result is then declared on the institute notice boards as well put on the web site and Students Corner. It is eventually sent to the University.

29 Who will keep the Student Academic Records, University or IARE?

It is the responsibility of the Examination Control Office of the institute to keep and preserve all the records.

30 What is our relationship with the JNT University?

We remain an affiliated college of the JNT University. The University has the right to nominate its members on the academic bodies of the college.

31 Shall we require University approval if we want to start any New Courses?

Yes, It is expected that approvals or such other matters from an autonomous college will receive priority.

32 Shall we get autonomy for PG and Doctoral Programs also?

Yes, presently our PG programs also enjoying autonomous status.

MALPRACTICE RULES

DISCIPLINARY ACTION FOR / IMPROPER CONDUCT IN EXAMINATIONS

S. No	Nature of Malpractices / Improper conduct	Punishment
	If the candidate:	
1. (a)	Possesses or keeps accessible in examination hall, any paper, note book, programmable calculator, cell phone, pager, palm computer or any other form of material concerned with or related to the course of the examination (theory or practical) in which he is appearing but has not made use of (material shall include any marks on the body of the candidate which can be used as an aid in the course of the examination)	Expulsion from the examination hall and cancellation of the performance in that course only.
(b)	Gives assistance or guidance or receives it from any other candidate orally or by any other body language methods or communicates through cell phones with any candidate or persons in or outside the exam hall in respect of any matter.	Expulsion from the examination hall and cancellation of the performance in that course only of all the candidates involved. In case of an outsider, he will be handed over to the police and a case is registered against him.
2.	Has copied in the examination hall from any paper, book, programmable calculators, palm computers or any other form of material relevant to the course of the examination (theory or practical) in which the candidate is appearing.	Expulsion from the examination hall and cancellation of the performance in that course and all other courses the candidate has already appeared including practical examinations and project work and shall not be permitted to appear for the remaining examinations of the courses of that Semester/year. The Hall Ticket of the candidate is to be cancelled and sent to the Controller of Examinations.
3.	Impersonates any other candidate in connection with the examination.	The candidate who has impersonated shall be expelled from examination hall. The candidate is also debarred and forfeits the seat. The performance of the original candidate, who has been impersonated, shall be cancelled in all the courses of the examination (including practicals and project work) already appeared and shall not be allowed to appear for examinations of the remaining courses of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is course to the academic regulations in connection with forfeiture of seat. If the imposter is an outsider, he will be handed over to the police and a case is registered against him.
4.	Smuggles in the Answer book or additional sheet or takes out or arranges to send out the question paper during the examination or answer book or additional sheet, during or after the examination.	Expulsion from the examination hall and cancellation of performance in that course and all the other courses the candidate has already appeared including practical examinations and project work and shall not be permitted for the

		remaining examinations of the courses of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is course to the academic regulations in connection with forfeiture of seat.
5.	Uses objectionable, abusive or offensive language in the answer paper or in letters to the examiners or writes to the examiner requesting him to award pass marks.	Cancellation of the performance in that course.
6.	Refuses to obey the orders of the Controller of Examinations /Additional Controller of Examinations/any officer on duty or misbehaves or creates disturbance of any kind in and around the examination hall or organizes a walk out or instigates others to walk out, or threatens the COE or any person on duty in or outside the examination hall of any injury to his person or to any of his relations whether by words, either spoken or written or by signs or by visible representation, assaults the COE or any person on duty in or outside the examination hall or any of his relations, or indulges in any other act of misconduct or mischief which result in damage to or destruction of property in the examination hall or any part of the Institute premises or engages in any other act which in the opinion of the officer on duty amounts to use of unfair means or misconduct or has the tendency to disrupt the orderly conduct of the examination.	In case of students of the college, they shall be expelled from examination halls and cancellation of their performance in that course and all other courses the candidate(s) has (have) already appeared and shall not be permitted to appear for the remaining examinations of the courses of that semester/year. The candidates also are debarred and forfeit their seats. In case of outsiders, they will be handed over to the police and a police case is registered against them.
7.	Leaves the exam hall taking away answer script or intentionally tears off the script or any part thereof inside or outside the examination hall.	Expulsion from the examination hall and cancellation of performance in that course and all the other courses the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the courses of that semester/year. The candidate is also debarred for two consecutive semesters from class work and all semester end examinations. The continuation of the course by the candidate is course to the academic regulations in connection with forfeiture of seat.
8.	Possess any lethal weapon or firearm in the examination hall.	Expulsion from the examination hall and cancellation of the performance in that course and all other courses the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the courses of that semester/year. The candidate is also debarred and forfeits the seat.
9.	If student of the college, who is not a candidate for the particular examination or any person not connected with the college indulges in any	Student of the colleges expulsion from the examination hall and cancellation of the performance in that course and all other

	malpractice or improper conduct mentioned in clause 6 to 8.	courses the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the courses of that semester/year. The candidate is also debarred and forfeits the seat. Person(s) who do not belong to the College will be handed over to police and, a police case will be registered against them.
10.	Comes in a drunken condition to the examination hall.	Expulsion from the examination hall and cancellation of the performance in that course and all other courses the candidate has already appeared including practical examinations and project work and shall not be permitted for the remaining examinations of the courses of that semester/year.
11.	Copying detected on the basis of internal evidence, such as, during valuation or during special scrutiny.	Cancellation of the performance in that course and all other courses the candidate has appeared including practical examinations and project work of that semester/year examinations.
12.	If any malpractice is detected which is not covered in the above clauses 1 to 11 shall be reported to the University for further action to award suitable punishment.	

FAILURE TO READ AND UNDERSTAND THE REGULATIONS IS NOT AN EXCUSE

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

COURSE CATALOGUE

REGULATIONS: BT-23

INFORMATION TECHNOLOGY

I SEMESTER

Course Code	Course Name	Subject Area	Category	Periods Per Week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
INDUCTION PROGRAM				**TWO WEEKS MANDATORY AUDIT COURSE**						
THEORY										
AHSD01	Professional Communication	HSMC	Foundation	3	0	0	3	40	60	100
AHSD02	Matrices and Calculus	BSC	Foundation	3	1	0	4	40	60	100
AEED01	Elements of Electrical and Electronics Engineering	ESC	Foundation	3	0	0	3	40	60	100
ACSD01	Object Oriented Programming	ESC	Foundation	3	0	0	3	40	60	100
PRACTICAL										
AHSD04	Professional Communication Laboratory	HSMC	Foundation	0	0	2	1	40	60	100
ACSD02	Object Oriented Programming with Java Laboratory	ESC	Foundation	0	1	2	2	40	60	100
AMED02	Manufacturing Practice	ESC	Foundation	1	0	2	2	40	60	100
AEED03	Electrical and Electronics Engineering Laboratory	ESC	Foundation	0	0	2	1	40	60	100
EXPERIENTIAL ENGINEERING EDUCATION (ExEED)										
ACSD03	Essentials of Innovation	Skill	Skill	0	0	2	1	40	60	100
MANDATORY COURSE										
AHSD06	Environmental Science	MC	MC - I	**Ref: Academic Regulations BT23**						
TOTAL				**13**	**02**	**10**	**20**	**360**	**540**	**900**

II SEMESTER

Course Code	Course Name	Subject Area	Category	Periods Per Week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSD03	Engineering Chemistry	BSC	Foundation	3	0	0	3	40	60	100
AHSD07	Applied Physics	BSC	Foundation	3	0	0	3	40	60	100
AHSD08	Differential Equations and Vector Calculus	BSC	Foundation	3	1	0	4	40	60	100
ACSD05	Essentials of Problem Solving	ESC	Foundation	3	0	0	3	40	60	100
PRACTICAL										
AHSD05	Engineering Chemistry Laboratory	BSC	Foundation	0	0	2	1	40	60	100
AHSD09	Applied Physics Laboratory	BSC	Foundation	0	0	2	1	40	60	100
ACSD06	Programming for Problem Solving Laboratory	ESC	Foundation	0	1	2	2	40	60	100
AMED03	Engineering Graphics	ESC	Foundation	1	0	2	2	40	60	100
SKILL ENHANCEMENT PROJECT										
ACSD04	Mobile Applications Development	Skill	Skill	0	0	2	1	40	60	100
MANDATORY COURSE										
AHSD10	Gender Sensitization	MC	MC - II	**Ref: Academic Regulations BT23**						
FIELD PROJECT										
TOTAL				**13**	**02**	**10**	**20**	**360**	**540**	**900**

III SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSD11	Probability and Statistics	PCC	Core	3	1	0	4	40	60	100
AECD04	Computer System Architecture	PCC	Core	3	0	0	3	40	60	100
ACSD08	Data Structures	PCC	Core	3	0	0	3	40	60	100
ACSD09	Operating Systems	PCC	Core	3	0	0	3	40	60	100
AITD01	Mathematics for Computing	PCC	Core	3	0	0	3	40	60	100
PRACTICAL										
ACSD10	Operating Systems Laboratory	PCC	Core	0	0	2	1	40	60	100
ACSD11	Data Structures Laboratory	PCC	Core	0	0	2	1	40	60	100
AITD02	Programming with Objects Laboratory	PCC	Core	0	0	2	1	40	60	100
EXPERIENTIAL ENGINEERING EDUCATION (ExEED)										
ACSD12	Prototype and Design Building	Skill	Skill	0	0	2	1	40	60	100
TOTAL				**15**	**01**	**08**	**20**	**360**	**540**	**900**

IV SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACSD13	Design and Analysis of Algorithms	PCC	Core	3	0	0	3	40	60	100
ACSD14	Web Systems Engineering	PCC	Core	3	0	0	3	40	60	100
ACSD15	Object Oriented Software Engineering	PCC	Core	3	0	0	3	40	60	100
AITD03	Database Management Systems	PCC	Core	3	0	0	3	40	60	100
AITD04	Computer Networks	PCC	Core	3	0	0	3	40	60	100
PRACTICAL										
ACSD16	Design and Analysis of Algorithms Laboratory	PCC	Core	0	0	2	1	40	60	100
ACSD17	Web Systems Engineering Laboratory	PCC	Core	0	0	2	1	40	60	100
AITD05	Database Management Systems Laboratory	PCC	Core	0	0	2	1	40	60	100
SKILL ENHANCEMENT PROJECT										
ACSD18	DevOps Engineering	Skill	Skill	1	0	2	2	40	60	100
VALUE ADDED COURSE										
INTERNSHIP										
TOTAL				**16**	**00**	**08**	**20**	**360**	**540**	**900**

#The course would consist of talks by working professionals from industry, government, academia & research organizations.

V SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACSD19	Data Mining and Knowledge Discovery	PCC	Core	3	0	0	3	40	60	100
ACCD04	Information Security Management	PCC	Core	3	0	0	3	40	60	100
ACSD20	Cloud Application Development	PCC	Core	3	0	0	3	40	60	100
ACSD21	Artificial Intelligence	PCC	Core	3	0	0	3	40	60	100
	Program Elective – I	PEC	Elective	3	0	0	3	40	60	100
PRACTICAL										
ACSD26	Artificial Intelligence Laboratory	PCC	Core	0	0	2	1	40	60	100
ACSD27	Cloud Application Development Laboratory	PCC	Core	0	0	2	1	40	60	100
SKILL ENHANCEMENT PROJECT										
ACSD29	Engineering Design Project	Skill	Skill	0	0	2	1	40	60	100
ACSD30	Java Full Stack Development	Skill	Skill	1	0	2	2	40	60	100
VALUE ADDED COURSE										
TOTAL				**16**	**-**	**08**	**20**	**360**	**540**	**900**

#The course would consist of talks by working professionals from industry, government, academia & research organizations.

VI SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
ACSD31	Theory of Computation	PCC	Core	3	0	0	3	40	60	100
ACSD32	Machine Learning and Neural Computing	PCC	Core	3	0	0	3	40	60	100
	Program Elective – II	PEC	Elective	3	0	0	3	40	60	100
	Program Elective – III	PEC	Elective	3	0	0	3	40	60	100
	Open Elective – I	OEC	Elective	3	0	0	3	40	60	100
PRACTICAL										
ACSD41	Computer System Internals and Linux Laboratory	PCC	Core	0	0	2	1	40	60	100
ACSD42	Neural Computing and Deep Learning Laboratory	PCC	Core	0	0	2	1	40	60	100
SKILL ENHANCEMENT PROJECT										
ACSD44	Data Scientist / AI Specialist	Skill	Skill	1	0	2	2	40	60	100
ACSD45	Development Project	Skill	Skill	0	0	2	1	40	60	100
VALUE ADDED COURSE										
INTERNSHIP										
TOTAL				**16**	**00**	**08**	**20**	**360**	**540**	**900**

#The course would consist of talks by working professionals from industry, government, academia & research organizations.

VII SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AITD19	Cyber Physical Systems	PCC	Core	3	0	0	3	40	60	100
ACSD46	Compiler Construction	PCC	Core	3	0	0	3	40	60	100
	Program Elective - IV	PEC	Elective	3	0	0	3	40	60	100
	Program Elective - V	PEC	Elective	3	0	0	3	40	60	100
	Open Elective - II	OEC	Elective	3	0	0	3	40	60	100
PRACTICAL										
ACSD54	Computing Things Laboratory	PCC	Core	0	0	2	1	40	60	100
ACSD55	Security Protocols Laboratory	PCC	Core	0	0	2	1	40	60	100
PROJECT WORK										
AITD27	Project Work (Phase - I)	PROJ	Project	0	0	6	3	40	60	100
MANDATORY COURSE										
AHSD14	Essence of Indian Traditional Knowledge	MC	MC-III	**Ref: Academic Regulations BT23**						
TOTAL				**15**	**00**	**10**	**20**	**320**	**480**	**800**

VIII SEMESTER

Course Code	Course Name	Subject Area	Category	Periods per week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSD15	Managerial Economics and Financial Analysis	BSC	Foundation	3	0	0	3	40	60	100
	Program Elective - VI	PEC	Elective	3	0	0	3	40	60	100
	Open Elective - III	OEC	Elective	3	0	0	3	40	60	100
PROJECT WORK										
AITD33	Project Work (Phase - II)	PCC	Project	0	0	22	11	40	60	100
TOTAL				**09**	**00**	**22**	**20**	**160**	**240**	**400**

ELECTIVE COURSES

PROGRAM ELECTIVES COURSES (PEC)

Course Code	Name of the Course	Prerequisites	Preferred Semester	Credits
AITD06	Advanced Data Structures and Algorithms	Data Structures	V	3
AITD07	Media Engineering	No prerequisite	V	3
AITD08	Introduction to Communication System	No Prerequisite	V	3
AITD09	Computer Graphics	Matrices and Calculus	V	3
ACSD24	Advanced Computer Architecture	Matrices and Calculus	V	3
ACCD05	Computer Vision	Probability and Statistics, Data Structures	VI	3
AITD10	Social Networks	Probability and Statistics, Data Structures	VI	3
ACSD34	Advanced Algorithms	Design and Analysis of Algorithms	VI	3
AITD11	Natural Language Processing	Probability and Statistics	VI	3
AITD12	Quantum Computing	Probability and Statistics	VI	3
AITD13	High Performance Computing	Probability and Statistics, Data Structures	VI	3
AITD14	Game Theory	Probability and Statistics	VI	3
AECD19	Microprocessor and Microcontrollers	Digital Design and Embedded Systems	VI	3
AITD15	Real Time Systems	Operating Systems	VI	3
AITD16	Agile Development and Scrum Practices	Object Oriented Software Engineering	VI	3
AITD20	Big Data - Tools and Techniques	Database Management Systems	VII	3
AITD21	Data Analytics	Artificial Intelligence	VII	3
AITD22	Software Testing Methodologies	Software Engineering	VII	3
ACSD25	Software Project Management	Software Engineering	VII	3
ACDD23	Reinforcement Learning	Artificial Intelligence	VII	3
AITD23	Multimedia Systems	Artificial Intelligence	VII	3
ACDD25	IoT Security and Attacks	Cyber Security	VII	3
AITD24	Network Automation	Computer Networks	VII	3
AITD25	Text Mining	Data Mining and Knowledge Discovery	VII	3
AITD26	Web Mining	Data Mining and Knowledge Discovery	VII	3
ACAD14	Human Computer Interaction (UI & UX)	Scripting Languages	VIII	3
ACSD57	Blockchain Technology	Computer Networks	VIII	3
AITD28	Computer Forensics	Information Security Management	VIII	3
AITD29	Ethical Hacking and Cyber Security	Information Security Management	VIII	3
AITD30	Cyber Security	Information Security Management	VIII	3

OPEN ELECTIVE COURSES (OEC)

The courses listed below are offered by the Department of Information Technology for students of other departments.

Course Code	Course Name	Prerequisites	Credits
AITD17	Cloud and Grid Computing	-	3
AITD18	Computer Graphics and Multimedia Systems	-	3
ACSD52	Bio-Informatics	-	3
ACSD53	Ethical Hacking	-	3
AITD31	E-Commerce	-	3
AITD32	Visualizations and Animations	-	3

VALUE-ADDED COURSES

Objective:

Value added courses are provided to equip the students with knowledge and skills outside of the curriculum or to meet any specific requirements of the industry. The following are the Value-Added Courses provided to students by various departments in our institution.

ANY THREE FROM THE GIVEN BELOW THROUGH IARE ELRV – AKANKSHA / CERTIFICATE - SWAYAM, e-PG pathshala, NPTEL etc..

1. Data Scalability and Distribution (Amazon Web Services, Microsoft Azure, Google Cloud Platform etc.
2. Software Developer (Restful webservices / Microservices, Rust programming, MEAN, MERN, MEVN)
3. Data Science (Data visualization, Data wrangling, Bigdata Technologies, Business Intelligence etc..)
4. Operating Systems (IBM I, Mac OS, Linux, Haiku etc.)
5. Debugging
6. Testing (Selenium, TestNG)
7. Cyber Security (Network Security, Threat Intelligence and Analysis / Risk Assessment and Management
8. Software Architect
9. Blockchain Technology
10. Creative Image Captioning Models

COURSE CONTENT

PROFESSIONAL COMMUNICATION								
I Semester: AE / ME / CE / CSE (AI &ML) / IT / ECE / EEE **II Semester:** CSE | CSE (DS) | CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AHSD01	**Foundation**	3	-	-	3	40	60	100
Contact Classes: 64	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisite:								

I. COURSE OVERVIEW:

The principle aim of the course is that the students will have awareness about the importance of English language in the contemporary times and also it emphasizes the students to learn this languageas a skill (listening skill, speaking skill, reading skill and writing skill). Moreover, the course benefitsthe students how to solve their day-to-day problems in speaking English language. Besides, it assiststhe students to reduce the mother tongue influence and acquire the knowledge of neutral accent. Thecourse provides theoretical and practical knowledge of English language and it enables students to participate in debates about informative, persuasive, didactic, and commercial purposes.

II. COURSE OBJECTIVES:

The students will try to learn:

I	Standard pronunciation, appropriate word stress, and necessary intonation patterns foreffective communication towards achieving academic and professional targets.
II	Appropriate grammatical structures and also using the nuances of punctuation tools forpractical purposes.
III	Critical aspect of speaking and reading for interpreting in-depth meaning between thesentences.
IV	Conceptual awareness on writing in terms of unity, content, coherence, and linguisticaccuracy.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO 1	Demonstrate the prime necessities of listening skills and communication skills for academic and non-academic purposes.
CO 2	Explain effectively in spoken English on issues and ideas with a reasonable degree of fluency and accuracy in different social settings and different kinds of social encounters.
CO 3	Strengthen acceptable language for developing life skills to overcome the challenges at professional platform.
CO 4	Interpret the grammatical and lexical forms of English and use of these forms in specific communicative contexts.
CO 5	Articulate main ideas, both stated and inferred, and important details in academic, journalistic, and literary prose at advanced level.
CO 6	Extend writing skills for fulfilling academic and work-place requirements of various written communicative functions.

IV.COURSE CONTENT:

MODULE – I: GENERAL INTRODUCTION AND LISTENING SKILLS (13)

Introduction to communication skills, communication process, elements of communication, soft skills and hard skills, importance of soft skills for engineers, listening skills, significance of listening skills, stages of listening, barriers and effectiveness of listening, listening comprehension.

MODULE – II: SPEAKING SKILL (13)

Significance of speaking skills, essentials of speaking skills, verbal and non-verbal communication, generating talks based on visual prompts, public speaking, exposure to structured talks, delivering speech effectively, oral presentation using power point slides.

MODULE – III: VOCABULARY AND GRAMMAR (13)
The concept of word formation, idioms and phrases, one-word substitutes, sentence structure (simple, compound and complex), usage of punctuation marks, advanced level prepositions.

Tenses, subject verb agreement, degrees of comparison, direct and indirect speech, questions tags.

MODULE – IV: READING SKILL (12)
Significance of reading skills, techniques of reading, skimming-reading for the gist of a text, scanning–reading for specific information, intensive, extensive reading, reading comprehension, metaphor and figurative language.

MODULE – V: WRITING SKILL (13)
Significance of writing skills, effectiveness of writing, the role of a topic sentence andsupporting sentences in a paragraph, organizing principles of paragraphs in a document, writing introduction and conclusion, techniques for writing precis, various formats for letter writing (block format, full block format, and semi bloc format), e-mail writing, report writing.

V. TEXT BOOKS:

1. Anjana Tiwari, *Communication Skills in English*, Khanna Publishing House: New Delhi,2022.

VI. REFERENCE BOOKS:

1. Norman Whitby, *Business Benchmark: Pre-Intermediate to Intermediate* – BECPreliminary, Cambridge University Press, 2nd Edition, 2008.
2. Devaki Reddy, Shreesh Chaudhary, *Technical English*, Macmillan, 1st Edition, 2009.
3. Rutherford, Andrea J, *Basic Communication Skills for Technology*, Pearson Education, 2ndEdition, 2010.
4. Raymond Murphy, *Essential English Grammar with Answers*, Cambridge UniversityPress, 2nd Edition, 2010.

VII. ELECTRONICS RESOURCES:

1. https://akanksha.iare.ac.in/index?route=course/details&course_id=954
2. https://akanksha.iare.ac.in/index?route=course/details&course_id=10
3. https://akanksha.iare.ac.in/index?route=course/details&course_id=352
4. https://akanksha.iareac.in/index?route=publicprofile&id=5075

VIII. MATERIALS ONLINE

1. Course template
2. Tech-talk topics
3. Assignments
4. Definition and terminology
5. Tutorial question bank
6. Model question paper – I
7. Model question paper – II
8. Lecture notes
9. Early lecture readiness videos (ELRV)
10. Power point presentations

COURSE CONTENT

MATRICES AND CALCULUS								
I Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours/Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AHSD02	Foundation	3	1	-	4	40	60	100
Contact Classes: 48	Tutorial Classes: 16	Practical Classes: Nil				Total Classes: 64		
Prerequisite: Basic Principles of Algebra and Calculus								

I. COURSE OVERVIEW:

This course Matrices and Calculus is a foundation course of mathematics for all engineering branches. The concepts of Matrices, Eigen Values, Eigen Vectors, Functions of Single and Several Variables, Fourier Series and Multiple Integrals. This course is applicable for simulations, colour imaging process, finding optimal solutions in all fields of industries.

II. COURSE OBJECTIVES:

The students will try to learn:

I	The concept of the rank of a matrix, solve the system of linear equations, eigen values, eigen vectors.
II	The geometrical approach to the mean value theorems and their application to the mathematical problems.
III	The Fourier series expansion in standard intervals as well as arbitrary intervals.
IV	The evaluation of multiple integrals and their applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Determine the rank and solutions of linear equations with elementary operations.
CO 2	Utilize the Eigen values, Eigen vectors for developing spectral matrices.
CO 3	Make use of Cayley-Hamilton theorem for finding powers of the matrix.
CO 4	Interpret the maxima and minima of given functions by finding the partial derivates.
CO 5	Apply the Fourier series expansion of periodic functions for analyzing the wave forms.
CO 6	Determine the area of solid bounded regions by using the integral calculus.

IV. COURSE CONTENT:

MODULE - I: MATRICES (09)

Rank of a matrix by echelon form and normal form, inverse of non-singular matrices by Gauss-Jordan method, system of linear equations, solving system of homogeneous and non-homogeneous equations.

MODULE - II: EIGEN VALUES AND EIGEN VECTORS (10)

Eigen values, eigen vectors and their properties (without proof), Cayley-Hamilton theorem (without proof), verification, finding inverse and power of a matrix by Cayley-Hamilton theorem, diagonalization of a matrix.

MODULE - III: FUNCTIONS OF SINGLE and SEVERAL VARIABLES (10)

Mean value theorems Rolle’s theorem, Lagrange’s theorem, Cauchy’s theorem-without proof.

Functions of several variables: Partial differentiation, Jacobian, functional dependence, maxima and minima of functions of two variables and three variables, method of Lagrange multipliers.

MODULE – IV: FOURIER SERIES (09)

Fourier expansion of periodic function in a given interval of length 2π, Fourier series of even and odd functions, Fourier series in an arbitrary interval, half- range Fourier sine and cosine expansions.

MODULE – V: MULTIPLE INTEGRALS (10)

Evaluation of double integrals (cartesian and polar coordinates), change of order of integration (only cartesian coordinates), evaluation of triple integrals (only cartesian coordinates).

V. TEXT BOOKS:

1. B. S. Grewal, *Higher Engineering Mathematics*, 44/e, Khanna Publishers, 2017.
2. Erwin Kreyszig, *Advanced Engineering Mathematics*, 10/e, John Wiley & Sons, 2011.

VI. REFERENCE BOOKS:

1. R. K. Jain and S. R. K. Iyengar, *Advanced Engineering Mathematics*, 3/ed, Narosa Publications, 5th Edition, 2016.
2. George B. Thomas, Maurice D. Weir and Joel Hass, Thomas, *Calculus*, 13/e, Pearson Publishers, 2013.
3. N.P. Bali and Manish Goyal, *A text book of Engineering Mathematics*, Laxmi Publications, Reprint, 2008.
4. Dean G. Duffy, *Advanced Engineering Mathematics with MATLAB*, CRC Press.
5. Peter O'Neil, *Advanced Engineering Mathematics*, Cengage Learning.
6. B.V. Ramana, *Higher Engineering Mathematics*, McGraw Hill Education.

VII. ELECTRONICS RESOURCES:

1. Engineering Mathematics - I, By Prof. Jitendra Kumar | IIT Kharagpur https://onlinecourses.nptel.ac.in/noc23_ma88/preview
2. Advanced Calculus for Engineers, By Prof. Jitendra Kumar, Prof. Somesh Kumar | IIT Kharagpur https://onlinecourses.nptel.ac.in/noc23_ma86/preview
3. http://www.efunda.com/math/math_home/math.cfm
4. http://www.ocw.mit.edu/resourcs/#Mathematics
5. http://www.sosmath.com
6. http://www.mathworld.wolfram.com

VIII. MATERIAL ONLINE:

1. Course template
2. Tech-talk topics
3. Assignments
4. Definition and terminology
5. Tutorial question bank
6. Model question paper – I
7. Model question paper – II
8. Lecture notes
9. Early lecture readiness videos (ELRV)
10. Power point presentations

COURSE CONTENT

ELEMENTS OF ELECTRICAL AND ELECTRONICS ENGINEERING								
I Semester: CSE (AI&ML) / IT / AERO / MECH / CIVIL **II Semester:** CSE / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AEED01	**Foundation**	3	-	-	3	40	60	100
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisite: Physics								

I. COURSE OVERVIEW:

This course enables knowledge on electrical quantities such as current, voltage, and power, energy to know the impact of technology in global and societal context. It provides the knowledge on basic DC and AC circuits used in electrical and electronic devices, highlights the importance of electrical machines and basics of semiconductor devices like diodes and transistors.

II. COURSES OBJECTIVES:

The students will try to learn

I	The fundamentals of electrical circuits and analysis of circuits with DC and AC excitation using circuit laws.
II	The construction and operation of Electrical machines.
III	The operational characteristics of semiconductor devices with their applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Make use of basic electrical laws for solving DC and AC circuits.
CO 2	Apply network theorems for analysis of simple electrical circuits.
CO 3	Demonstrate the fundamentals of electromagnetism for the operation of DC and AC machines.
CO 4	Utilize the characteristics of semiconductor devices for the application of rectifiers and regulators.
CO 5	Interpret the transistor configurations for optimization of the operating point.
CO 6	Understand the amplifier circuits using transistors for calculating different parameters

IV. COURSE CONTENT:

MODULE-I: INTRODUCTION TO ELECTRICAL CIRCUITS (09)

Circuit concept: Ohm's law, Kirchhoff's laws, the equivalent resistance of networks, star to delta transformation, mesh and nodal analysis (with DC source only).

Single phase AC circuits: representation of alternating quantities, RMS, average, form and peak factor, RLC series circuit.

MODULE-II: NETWORK THEOREMS AND THREE-PHASE VOLTAGES (10)

Network Theorems: Superposition, reciprocity, Thevenin's, Norton's, Maximum power transfer theorems for DC excitation circuits, three phase voltages (Definitions only): Voltage and current relationships in star and delta connections.

MODULE-III: ELECTRICAL MACHINES AND SEMICONDUCTOR DIODES (10)

DC and AC machines: Motors and generators, principle of operation, parts, EMF equation, types, applications, losses and efficiency.

Semiconductor diode: P-N Junction diode, symbol, V-I characteristics, half wave rectifier, full wave rectifier, bridge rectifier and filters, diode as a switch, Zener diode as a voltage regulator.

MODULE-IV: BIPOLAR JUNCTION TRANSISTOR AND APPLICATIONS (10)
Bipolar junction transistor: Characteristics and configurations, working principle NPN and PNP transistor, CE, CB, CC configurations – input and output characteristics, transistor as a switch.

MODULE-V: TRANSISTOR AMPLIFIERS (09)
Amplifier circuits: Two port devices and network, small signal models for transistors, concept of small signal operation, amplification in CE amplifier, h parameter model of a BJT- CE, CB and emitter follower analysis.

V. TEXT BOOKS:

1. M. S. Sukhija, T. K. Nagsarkar, *Basic Electrical and Electronics Engineering*, Oxford, 1st Edition, 2012.
2. Salivahanan, *Electronics Devices & Circuits*, TMH 4th Edition 2012.

VI. REFERENCE BOOKS:

1. CL Wadhwa, *Electrical Circuit Analysis including Passive Network Synthesis*, International, 2nd Edition, 2009.
2. David A Bell, *Electric circuits*, Oxford University Press,7th Edition,2009.
3. PS Bimbra, *Electrical Machines*, Khanna Publishers, 2nd Edition,2008.
4. D.P. Kothari and I. J. Nagrath, *Basic Electrical Engineering*, Tata McGraw Hill, 4th Edition, 2019.

VII. ELECTRONICS RESOURCES:

1. https://www.kuet.ac.bd/webportal/ppmv2/uploads/1364120248DC%20Machines
2. https://www.eleccompengineering.files.wordpress.com/2014/08/a-textbook-of-electrical-technologyvolume-ii-ac-and-dc-machines-b-l-thferaja.pdf
3. https://www.geosci.uchicago.edu/~moyer/GEOS24705/Readings/Klempner_Ch1.pdf
4. https://www.ibiblio.org/kuphaldt/electricCircuits/DC/DC.pdf
5. https://www.users.ece.cmu.edu/~dwg/personal/sample.pdf.
6. https://www.iare.ac.in

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Definitions and terminology
5. Open ended experiments
6. Model question paper-i
7. Model question paper-ii
8. Lecture notes
9. Early learning readiness videos (elrv)
10. Power point presentations

COURSE CONTENT

OBJECT ORIENTED PROGRAMMING								
I Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD01	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisites: **There are no prerequisites to take this course.**								

I. COURSE OVERVIEW:

The course provides a solid foundation in object-oriented programming concepts in using them. It includes concepts object-oriented concepts such as information hiding, encapsulation, and polymorphism. It contrasts the use of inheritance and composition as techniques for software reuse. It provides an understanding of object-oriented design using graphical design notations such as Unified Modeling Language (UML) as well as object design patterns.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts and principles of object-oriented programming in high-level programming languages.
II. The advanced concepts for developing well-structured and efficient programs that involve complex data structures, numerical computations, or domain-specific operations.
III. The design and implementation of features such as inheritance, polymorphism, and encapsulation for tackling complex problems and creating well-organized, modular, and maintainable code.
IV. The usage of input/output interfaces to transmit and receive data to solve real-time computing problems.

III. COURSE OUTCOMES:

At the end of the course, students should be able to:

CO 1	Interpret the features of object-oriented programming languages, comparison, and evaluation of programming languages.
CO 2	Model the real-world scenario using class diagrams and exhibit communication between objects.
CO 3	Estimate the need for special functions for data initialization.
CO 4	Outline the features of object-oriented programming for binding the attributes and behavior of a real-world entity.
CO 5	Use the concepts of streams and files that enable data management to enhance programming skills.
CO 6	Develop contemporary solutions to software design problems using object-oriented principles.

IV. COURSE CONTENT:

MODULE - I: Object-oriented concepts (09)

Objects and legacy systems, procedural versus Object-oriented programming, top-down and bottom-up approaches and their differences, benefits of OOP, applications of OOP, features of OOP.

Abstraction: Layers of abstraction, forms of abstraction, abstraction mechanisms.

MODULE - II: Classes and objects (09)

Classes and objects: Object data, object behaviors, creating objects, attributes, methods, messages, creating class diagrams.

Access specifiers and initialization of class members: Accessing members and methods, access specifiers - public, private, protected, memory allocation. Static members, static methods.

MODULE - III: Special member functions and overloading (09)

Constructors and destructors: Need for constructors and destructors, copy constructors, dynamic constructors, parameterized constructors, destructors, constructors and destructors with static members.

Overloading: Function overloading, constructor overloading, and operator overloading - rules for overloading operators, overloading unary and binary operators, friend functions.

MODULE – IV: Inheritance and polymorphism (09)
Inheritance: types of inheritance, base class, derived class, usage of final, ambiguity in multiple and multipath inheritance, virtual base class, overriding member functions, order of execution of constructors and destructors.

Polymorphism and virtual functions: Virtual functions, pure virtual functions, abstract classes, introduction to polymorphism, static polymorphism, dynamic polymorphism.

MODULE –V: Console I/O and working with files (09)
Console I/O: Concept of streams, hierarchy of console stream classes, unformatted I/O operations, managing output with manipulators.

Working with files: Opening, reading, writing, appending, processing, and closing different types of files, command line arguments**.**

V. TEXTBOOKS:

1. Matt Weisfeld, *The Object-Oriented Thought Process*, Addison Wesley Object Technology Series, 4th Edition, 2013.

VI. REFERENCE BOOKS:

1. Timothy Budd, *Introduction to object-oriented programming*, Addison Wesley Object Technology Series, 3rd Edition, 2002.
2. Gaston C. Hillar, *Learning Object-Oriented Programming*, Packt Publishing, 2015.
3. Kingsley Sage, *Concise Guide to Object-Oriented Programming*, Springer International Publishing, 1st Edition, 2019.
4. Rudolf Pecinovsky**,** *OOP - Learn Object Oriented Thinking and Programming*, Tomas Bruckner, 2013.
5. Grady Booch, *Object-oriented analysis and design with applications*, Addison Wesley Object Technology Series, 3rd Edition, 2007.

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/javase/tutorial/java/concepts/
2. https://www.w3schools.com/cpp/
3. https://www.edx.org/learn/object-oriented-programming/
4. https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

PROFESSIONAL COMMUNICATION LABORATORY								
I Semester: CSE (AI&ML) | IT | AE | ECE | EEE | ME |CE | **II Semester:** CSE | CSE(DS) | CSE(CS)								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AHSD04	Foundation	L	T	P	C	CIA	SEE	Total
		-	-	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 45				Total Classes: 45		
Prerequisite: Nil								

I. COURSE OVERVIEW:

This laboratory course is designed to introduce students to create a wide exposure on language learning techniques of the basic elements of listening skills, speaking skills, reading skills and writing skills. In this laboratory, students are trained in communicative English language skills, phonetics, word accent, word stress, rhythm, intonation, oral presentations and extempore speeches. Students are also taught in terms of seminars, group-discussions, presenting techniques of writing, participating in role plays, telephonic etiquettes, asking and giving directions, information transfer, debates, description of persons, places and objects etc. The laboratory encourages students to work in a group, engage in peer-reviews and inculcate team spirit through various exercises on grammar, vocabulary, and pronunciation games etc. Students will make use of all these language skills in academic, professional and real time situations.

II. COURSES OBJECTIVES

The students will try to learn:

I. English speech sounds, word accent, intonation and stress patterns for effective pronunciation.
II. Critical aspect of speaking and reading for interpreting in-depth meaning between the sentences.
III. Language techniques for social interactions such as public speaking, group discussions and interviews.
IV. Computer-assisted multi-media instructions and independent language learning.

III. COURSE OUTCOMES:

At the end of the course, students will be able to:

CO 1	Articulate acceptable accent in order to execute formal and informal communication.
CO 2	Differentiate stress shifts, syllabification and make use of past tense and plural markers effectively in connected speech; besides participate in role plays with confidence.
CO 3	Use weak forms and strong forms in spoken language and maintain intonation patterns as a native speaker to avoid mother tongue influence; moreover, practice various etiquettes at professional platform.
CO 4	Demonstrate errors in pronunciation and the decorum of oral presentations; for that reason, take part joining in group discussions and debates with much critical observations.
CO 5	Strengthen writing effective messages, notices, summaries and also able to write reviews very critically of art and academical videos.
CO 6	Argue scholarly, giving the counters to open ended experiments, and also writing slogans for the products talentedly.

Dos

1. Turn up in a neat and formal dress code regularly and maintain punctuality.
2. Bring observation books and worksheets for every laboratory session without fail.
3. Keep lab record book up to date.
4. Students must adhere to the acceptable use of ICT resources policy.
5. CD ROM 's, USB and other multimedia equipment are for college use only.
6. Replace headsets onto the monitor and rearrange your chairs back into their position as you leave laboratory.
7. Get your lab worksheets evaluated and upload online within the stipulated time for online evaluation by the faculty concerned.
8. Conduct yourself at the best to be a good learner.

Don'ts

1. Do not use the on/off switch to reboot the system.
2. Do not breach copyright regulations.
3. The not install or download any software or modify or delete any system files on any laboratory computer.
4. Do not read or modify other users' files.
5. Do not damage, remove, or disconnect any labels, parts, cables or equipment.
6. The not make undue noise in the laboratories. Be considerate of the other lab users- this is study area.
7. No food and the beverages are allowed into computer laboratories.

IV.COURSE CONTENT

Exercises for professional communication laboratory

Note: Students are encouraged to bring their own laptops for laboratory practice session

Getting started exercises

1.Introduction to pronunciation

Experiments
CALL LAB: Introduction to pronunciation
ICS LAB: Introducing self and introducing others and feedback

Activities
a) Pronunciation practice among groups.
b) Follow formal rules of introducing self and others and giving the constructive feedback.
c) Pronunciation practice from the K-Van software

2. Introduction to phonetics

Experiments
CALL LAB: Introduction to phonetics, listening to English sounds, Vowel and Consonant sounds.
ICS LAB: Describing a person or place or a thing using relevant adjectives – feedback

Activities
a) Introduction to phonetics, listening to English sounds, Vowel and Consonant sounds
b) Describing a person or place or a thing using relevant adjectives and giving valid feedback.
c) Listening English phonemes from K-Van software

3. Syllabification of different lexemes

Experiments
CALL LAB: Structure of syllables.
ICS LAB: JAM Sessions using public address system

Activities
a) Participating in just a minute session
b) Practicing consonant clusters
c) Practicing different methods of dividing the syllables

4. Word accent and stress shift patterns

Experiments:
CALL LAB: Word accent and stress shifts.
ICS LAB: Asking for directions and giving directions

Activities:
a) Usage of appropriate lexical tools at the time of giving directions
b) Pronounce weak and strong forms using strategies in spoken language

5. Usage of markers in pronunciation

Experiments:

CALL LAB: Past tense and plural markers.
ICS LAB: Role plays on fixed expressions in various situations

Activities:

a) Addition of suffixes to verbs
b) Multiple exercises on nouns based its number
c) Execution of body language in different contexts

6. Use of form in connected speech

CALL LAB: Weak forms and strong forms
ICS LAB: Extempore-Picture

Activities:

a) Practice on recognition of objects
b) Preparation on connected speech based on tone boundaries
c) Execution while describing pictures

7. Intonation patterns

Experiments
CALL LAB: Intonation
ICS LAB: Interpretation of Proverbs and Idioms

Activities

a) Voce modulation in speech as per context
b) Application of proverbs in real life contexts
c) Exercises on idiomatic expressions

8. Neutralization of Mother Tongue influence (MTI)

Experiments
CALL LAB: Neutralization of Mother Tongue Influence (MTI).
ICS LAB: Etiquette

Activities

a) Mirror practicing to get rid of mother tongue influence
b) Listening to English speeches from native speakers
c) Etiquettes for social and professional behavior

9. Oral presentations

Experiments
CALL LAB: Common errors in pronunciation practice through tongue twisters.
ICS LAB: Oral Presentations

Activities

a) Practicing tongue twisters to accelerate language fluency
b) Practice on how to grab attention of audience
c) Formulating strategies for structured presentation

10. Minimal pairs and debates

Experiments
CALL LAB: Minimal pairs.
ICS LAB: Debates

Activities

a) Practicing minimal pair dominoes
b) Working on minimal pair-cards and counters
a) Executing interpersonal skills with argumentative issues

11. Listening comprehension skills

Experiments
CALL LAB: Listening comprehension.

ICS LAB: Group discussion

Activities
- a) Listening to dialogues of native speakers
- b) Listening and writing short stories
- b) Exercising group discussion on various social issues

12. Enriching writing skills

Experiments
CALL LAB: Demonstration on how to write leaflets, messages and notices.
ICS LAB: Techniques and methods to write summaries and reviews of videos

Activities
- a) Practice on writing suitable messages and notices
- b) Summary writing techniques
- c) Practice on writing reviews for sociopolitical videos

13. Activities on pronunciation skills

Experiments
CALL LAB: Pronunciation practice.
ICS LAB: Information transfer

Activities
- a) Exercise on commonly mispronounced words
- b) Preparation on similar words pronunciation
- c) Practice on transferring information through chats and diagrams

14. Writing reviews after watching statements

Experiments
CALL LAB: Open ended experiments-phonetics practice.
ICS LAB: Providing reviews and remarks

Activities
- a) Writing three term labels for vowels
- b) Writing three term labels for consonants
- c) Practice on giving reviews and remarks

15. Experiments on text to speech and writing slogans

Experiments
CALL LAB: Open-ended experiments-text to speech.
ICS LAB: Writing slogan related to the image

Activities
- a) Practice through streaming video lessons
- b) Training on creating dialogues and stories
- c) Practice on designing and writing slogans

V. TEXT BOOKS:

1. Professional Communication laboratory manual

VI. REFERENCE BOOK

1. Meenakshi Raman, Sangeetha Sharma, *Technical Communication Principles and Practices*, Oxford University Press, New Delhi, 3rd Edition, 2015.
2. Rhirdion, Daniel, *Technical Communication*, Cengage Learning, New Delhi, 1st Edition, 2009.

VII. ELECTRONICS RESOURCES

1. Cambridge online pronunciation dictionary https://dictionary.cambridge.org/
2. Fluentu website https://www.fluentu.com/

3. Repeat after us https://brycs.org/clearinghouse/3018/
4. Language lab https://brycs.org/clearinghouse/3018/
5. Oxford online videos

VIII. MATERIALS ONLINE

1. Course template
2. Lab manual

COURSE CONTENT

OBJECT ORIENTED PROGRAMMING WITH JAVA LABORATORY								
I Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD02	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	1	2	2	40	60	100
Contact Classes: 15	**Tutorial Classes: NIL**	**Practical Classes: 30**				**Total Classes: 45**		
Prerequisite: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

This course provides a solid foundation in object-oriented programming concepts and hands-on experience in using them. It introduces the concepts of abstraction and reusable code design via the object-oriented paradigm. Through a series of examples and exercises students gain coding skills and develop an understanding of professional programming practices. Mastering Java facilitate the learning of other technologies.

II. COURSES OBJECTIVES:

The students will try to learn

I. The strong foundation with the Java Virtual Machine, its concepts and features.
II. The systematic understanding of key aspects of the Java Class Library
III. The usage of a modern IDE with an object oriented programming language to develop programs.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Develop non-trivial programs in an modern programming language.
CO 2	Apply the principles of selection and iteration.
CO 3	Appreciate uses of modular programming concepts for handling complex problems.
CO 4	Recognize and apply principle features of object-oriented design such as abstraction and encapsulation.
CO 5	Design classes with a view of flexibility and reusability.
CO 6	Code, test and evaluate small use cases to conform to a specification.

IV. COURE CONTENT:

EXERCISES FOR OBJECT ORIENTED PROGRAMMING WITH JAVA LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 HelloWorld

1. Install JDK on your machine.
2. Write a Hello-world program using JDK and a source-code editor, such as:
   - For All Platforms: Sublime Text, Atom
   - For Windows: TextPad, NotePad++
   - For macOS: jEdit, gedit
   - For Ubuntu: gedit
3. Do ALL the exercises.

1.2 Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.

It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these good programs. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.

Pay particular attention to:

1. Coding Style:
   - Read "Java Code Convention" (@ https://www.oracle.com/technetwork/java/codeconventions-150003.pdf or google "Java Code Convention").
   - Follow the Java Naming Conventions for variables, methods, and classes STRICTLY. Use CamelCase for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., radius) and class names (e.g., Circle). Use verbs for methods (e.g., getArea(), isEmpty()).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like i, j, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and i for array index. Use meaningful names like row and col (instead of x and y, i and j, x1 and x2), numStudents (not n), maxGrade, size (not n),
     and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use books for an array of books, and book for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse/NetBeans) can re-format your source codes with a single click.
2. Program Documentation: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.

1.3 Check Pass Fail (if-else)

Write a program called **CheckPassFail** which prints "PASS" if the int variable "mark" is more than or equal to 50; or prints "FAIL" otherwise. The program shall always print “DONE” before exiting.

Hints

Use >= for greater than or equal to comparison.

```
 /* Trying if-else statement.
 */
public class CheckPassFail {  // Save as "CheckPassFail.java"
   public static void main(String[] args) {  // Program entry point
      int mark = 49;   // Set the value of "mark" here!
      System.out.println("The mark is " + mark);
      // if-else statement
      if ( ...... ) {
         System.out.println( ...... );
      } else {
         System.out.println( ...... );
      }
      System.out.println( ...... );
   }
}
```

Try

mark = 0, 49, 50, 51, 100 and verify your results.

Take note of the source-code **indentation**!!! Whenever you open a block with '{', indent all the statements inside the block by 3 (or 4 spaces). When the block ends, un-indent the closing '}' to align with the opening statement.

1.4 CheckOddEven (if-else)

Write a program called **CheckOddEven** which prints "Odd Number" if the int variable “number” is odd, or “Even Number” otherwise. The program shall always print “bye!” before exiting.

Hints

n is an even number if (n % 2) is 0; otherwise, it is an odd number. Use == for comparison, e.g., (n % 2) == 0.

```
/**
 * Trying if-else statement and modulus (%) operator.
 */
public class CheckOddEven {   // Save as "CheckOddEven.java"
   public static void main(String[] args) {  // Program entry point
      int number = 49;       // Set the value of "number" here!
      System.out.println("The number is " + number);
      if ( ...... ) {
         System.out.println( ...... );   // even number
      } else {
         System.out.println( ...... );   // odd number
      }
      System.out.println( ...... );
   }
}
```

Try

number = 0, 1, 88, 99, -1, -2 and verify your results.

Again, take note of the source-code indentation! Make it a good habit to ident your code properly, for ease of reading your program.

1.5 PrintNumberInWord (nested-if, switch-case)

Write a program called **PrintNumberInWord** which prints "ONE", "TWO",... , "NINE", "OTHER" if the int variable "number" is 1, 2,... , 9, or other, respectively. Use (a) a "nested-if" statement; (b) a "switch-case-default" statement.

Hints

```
/**
 * Trying nested-if and switch-case statements.
 */
public class PrintNumberInWord {   // Save as "PrintNumberInWord.java"
   public static void main(String[] args) {
      int number = 5;  // Set the value of "number" here!
      // Using nested-if
      if (number == 1) {   // Use == for comparison
        System.out.println( ...... );
      } else if ( ...... ) {
        ......
      } else if ( ...... ) {
        ......
      ......
      ......
      } else {
        ......
      }

      // Using switch-case-default
      switch(number) {
        case 1:
          System.out.println( ...... ); break;  // Don't forget the "break" after each case!
        case 2:
          System.out.println( ...... ); break;
        ......
        ......
        default: System.out.println( ...... );
      }
    }
}
```

Try
number = 0, 1, 2, 3, ..., 9, 10 and verify your results.

1.6 PrintDayInWord (nested-if, switch-case)

Write a program called **PrintDayInWord** which prints "Sunday", "Monday", ... "Saturday" if the int variable "dayNumber" is 0, 1, ..., 6, respectively. Otherwise, it shall print "Not a valid day". Use (a) a "nested-if" statement; (b) a "switch-case-default" statement.

Try

dayNumber = 0, 1, 2, 3, 4, 5, 6, 7 and verify your results.

2. Exercises on Number Systems (for Science/Engineering Students)

To be proficient in programming, you need to be able to operate on these number systems:

1. Decimal (used by human beings for input and output)
2. Binary (used by computer for storage and processing)
3. Hexadecimal (shorthand or compact form for binary)

2.1 Exercises (Number Systems Conversion)

1. Convert the following decimal numbers into binary and hexadecimal numbers:
   a. 108
   b. 4848
   c. 9000

   Convert the following binary numbers into hexadecimal and decimal numbers:

   a. 10000000
   b. 101010101010
   c. 1000011000

   Convert the following hexadecimal numbers into binary and decimal numbers:

   a. 1234
   b. 80F

c. ABCDE

Convert the following decimal numbers into binary equivalent:

a. 123.456D
b. 19.25D

2.2 Exercise (Integer Representation)

1. What are the ranges of 8-bit, 16-bit, 32-bit and 64-bit integer, in "unsigned" and "signed" representation?
2. Give the value of 88, 0, 1, 127, and 255 in 8-bit unsigned representation.
3. Give the value of +88, -88 , -1, 0, +1, -128, and +127 in 8-bit 2's complement signed representation.
4. Give the value of +88, -88 , -1, 0, +1, -127, and +127 in 8-bit sign-magnitude representation.
5. Give the value of +88, -88 , -1, 0, +1, -127 and +127 in 8-bit 1's complement representation.

2.3 Exercises (Floating-point Numbers)

1. Compute the largest and smallest positive numbers that can be represented in the 32-bit normalized form.
2. Compute the largest and smallest negative numbers can be represented in the 32-bit normalized form.
3. Repeat (1) for the 32-bit denormalized form.
4. Repeat (2) for the 32-bit denormalized form.

Hints:

1. Largest positive number: S=0, E=1111 1110 (254), F=111 1111 1111 1111 1111 1111. Smallest positive number: S=0, E=0000 00001 (1), F=000 0000 0000 0000 0000 0000.
2. Same as above, but S=1.
3. Largest positive number: S=0, E=0, F=111 1111 1111 1111 1111 1111. Smallest positive number: S=0, E=0, F=000 0000 0000 0000 0000 0001.
4. Same as above, but S=1.

2.4 Exercises (Data Representation)

For the following 16-bit codes:

```
0000 0000 0010 1010;
1000 0000 0010 1010;
```

Give their values, if they are representing:

1. a 16-bit unsigned integer;
2. a 16-bit signed integer;
3. two 8-bit unsigned integers;
4. two 8-bit signed integers;
5. a 16-bit Unicode characters;
6. two 8-bit ISO-8859-1 characters.

3. Exercises on Decision and Loop

3.1 SumAverageRunningInt (Decision & Loop)

Write a program called **SumAverageRunningInt** to produce the sum of 1, 2, 3, ..., to 100. Store 1 and 100 in variables lowerbound and upperbound, so that we can change their values easily. Also compute and display the average.

The output shall look like:

```
The sum of 1 to 100 is 5050
The average is 50.5
```

Hints

```
/**
 * Compute the sum and average of running integers from a lowerbound to an upperbound using loop.
 */
public class SumAverageRunningInt {   // Save as "SumAverageRunningInt.java"
   public static void main (String[] args) {
      // Define variables
      int sum = 0;          // The accumulated sum, init to 0
      double average;       // average in double
      final int LOWERBOUND = 1;
      final int UPPERBOUND = 100;

      // Use a for-loop to sum from lowerbound to upperbound
      for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
            // The loop index variable number = 1, 2, 3, ..., 99, 100
         sum += number;     // same as "sum = sum + number"
      }
      // Compute average in double. Beware that int / int produces int!
      ......
      // Print sum and average
      ......
   }
}
```

Try

1. **Modify the program to use a "while-do" loop instead of "for" loop.**

```
int sum = 0;
      int number = LOWERBOUND;       // declare and init loop index variable
      while (number <= UPPERBOUND) {  // test
         sum += number;
         ++number;                    // update
      }
```

2. Modify the program using do-while loop

```
int sum = 0;
      int number = LOWERBOUND;        // declare and init loop index variable
      do {
         sum += number;
         ++number;                    // update
      } while (number <= UPPERBOUND);  // test
```

3. What is the difference between "for" and "while-do" loops? What is the difference between "while-do" and "do-while" loops?
4. Modify the program to sum from 111 to 8899, and compute the average. Introduce an int variable called count to count the numbers in the specified range (to be used in computing the average).
5. Modify the program to find the "sum of the squares" of all the numbers from 1 to 100, i.e. 1*1 + 2*2 + 3*3 + ... + 100*100.
6. Modify the program to produce two sums: sum of odd numbers and sum of even numbers from 1 to 100. Also computer their absolute difference.

3.2 Product1ToN (or Factorial) (Decision & Loop)

Write a program called Product1ToN to compute the product of integers from 1 to 10 (i.e., 1×2×3×...×10), as an int. Take note that It is the same as factorial of N.

Hints

Declare an int variable called product, initialize to 1, to accumulate the product.

```
    // Define variables
    int product = 1;      // The accumulated product, init to 1
    final int LOWERBOUND = 1;
    final int UPPERBOUND = 10;
```

Try

1. Compute the product from 1 to 11, 1 to 12, 1 to 13 and 1 to 14. Write down the product obtained and decide if the results are correct.

   HINTS: Factorial of 13 (=6227020800) is outside the range of int [-2147483648, 2147483647]. Take note that computer programs may not produce the correct result even though the code seems correct!
2. Repeat the above, but use long to store the product. Compare the products obtained with int for N=13 and N=14.

   HINTS: With long, you can store factorial of up to 20.

3.3 HarmonicSum (Decision & Loop)

Write a program called **HarmonicSum** to compute the sum of a harmonic series, as shown below, where n=50000. The program shall compute the sum from left-to-right as well as from the right-to-left. Are the two sums the same? Obtain the absolute difference between these two sums and explain the difference. Which sum is more accurate?

$$Harmonic(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Hints

```
/**
 * Compute the sum of harmonics series from left-to-right and right-to-left.
 */
public class HarmonicSum {   // Save as "HarmonicSum.java"
   public static void main (String[] args) {
      // Define variables
      final int MAX_DENOMINATOR = 50000;  // Use a more meaningful name instead of n
      double sumL2R = 0.0;          // Sum from left-to-right
      double sumR2L = 0.0;          // Sum from right-to-left
      double absDiff;             // Absolute difference between the two sums

      // for-loop for summing from left-to-right
      for (int denominator = 1; denominator <= MAX_DENOMINATOR; ++denominator) {
         // denominator = 1, 2, 3, 4, 5, ..., MAX_DENOMINATOR
         ......
         // Beware that int/int gives int, e.g., 1/2 gives 0.
      }
      System.out.println("The sum from left-to-right is: " + sumL2R);

      // for-loop for summing from right-to-left
      ......

      // Find the absolute difference and display
      if (sumL2R > sumR2L) ......
      else ......
   }
}
```

3.4 ComputePI (Decision & Loop)

Write a program called **ComputePI** to compute the value of π, using the following series expansion. Use the maximum denominator (MAX_DENOMINATOR) as the terminating condition. Try MAX_DENOMINATOR of 1000, 10000, 100000, 1000000 and compare the PI obtained. Is this series suitable for computing PI? Why?

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \cdots\right)$$

Hints

Add to sum if the denominator % 4 is 1, and subtract from sum if it is 3.

```
    double sum = 0.0;
    int MAX_DENOMINATOR = 1000;   // Try 10000, 100000, 1000000
    for (int denominator = 1; denominator <= MAX_DENOMINATOR; denominator += 2) {
        // denominator = 1, 3, 5, 7, ..., MAX_DENOMINATOR
      if (denominator % 4 == 1) {
        sum += ......;
      } else if (denominator % 4 == 3) {
        sum -= ......;
      } else {   // remainder of 0 or 2
        System.out.println("Impossible!!!");
      }
    }
    ......
```

Try

1. Instead of using maximum denominator as the terminating condition, rewrite your program to use the maximum number of terms (MAX_TERM) as the terminating condition.

```
int MAX_TERM = 10000;  // number of terms used in computation
    int sum = 0.0;
    for (int term = 1; term <= MAX_TERM; term++) {
        // term = 1, 2, 3, 4, ..., MAX_TERM
      if (term % 2 == 1) {  // odd term number: add
        sum += 1.0 / (term * 2 - 1);
      } else {              // even term number: subtract
        ......
      }
    }
```

2. JDK maintains the value of π in a built-in double constant called Math.PI (=3.141592653589793). Add a statement to compare the values obtained and the Math.PI, in percents of Math.PI, i.e., (piComputed / Math.PI) * 100.

3.5 CozaLozaWoza (Decision & Loop)

Write a program called **CozaLozaWoza** which prints the numbers 1 to 110, 11 numbers per line. The program shall print "Coza" in place of the numbers which are multiples of 3, "Loza" for multiples of 5, "Woza" for multiples of 7, "CozaLoza" for multiples of 3 and 5, and so on. The output shall look like:

```
1 2 Coza 4 Loza Coza Woza 8 Coza Loza 11
Coza 13 Woza CozaLoza 16 17 Coza 19 Loza CozaWoza 22
23 Coza Loza 26 Coza Woza 29 CozaLoza 31 32 Coza
......
```

Hints

```
public class CozaLozaWoza {   // Save as "CozaLozaWoza.java"
  public static void main(String[] args) {
    final int LOWERBOUND = 1, UPPERBOUND = 110;
    for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
        // number = LOWERBOUND+1, LOWERBOUND+2, ..., UPPERBOUND
      // Print "Coza" if number is divisible by 3
```

```
        if ( ...... ) {
          System.out.print("Coza");
        }
        // Print "Loza" if number is divisible by 5
        if ( ...... ) {
          System.out.print(.....);
        }
        // Print "Woza" if number is divisible by 7
        ......
        // Print the number if it is not divisible by 3, 5 and 7 (i.e., it has not been processed above)
        if ( ...... ) {
          ......
        }
        // After processing the number, print a newline if number is divisible by 11;
        // else print a space
        if ( ...... ) {
          System.out.println();  // print newline
        } else {
          System.out.print( ...... );  // print a space
        }
      }
    }
  }
```

Notes

1. You cannot use nested-if (if ... else if ... else if ... else) for this problem. It is because the tests are not mutually exclusive. For example, 15 is divisible by both 3 and 5. Nested-if is only applicable if the tests are mutually exclusive.

2. The tests above look messy. A better solution is to use a boolean flag to keep track of whether the number has been processed, as follows:

```
      final int LOWERBOUND = 1, UPPERBOUND = 110;
      boolean printed;
      for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
        printed = false;  // init before processing each number
        // Print "Coza" if number is divisible by 3
        if ( ...... ) {
          System.out.print( ...... );
          printed = true;   // processed!
        }
        // Print "Loza" if number is divisible by 5
        if ( ...... ) {
          System.out.print( ..... );
          printed = true;   // processed!
        }
        // Print "Woza" if number is divisible by 7
        ......
        // Print the number if it has not been processed
        if (!printed) {
          ......
        }
        // After processing the number, print a newline if it is divisible by 11;
        // else, print a space
        ......
      }
```

3.6 Fibonacci (Decision & Loop)

Write a program called **Fibonacci** to print the first 20 Fibonacci numbers F(n), where F(n)=F(n–1)+F(n–2) and F(1)=F(2)=1. Also compute their average. The output shall look like:

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
The average is 885.5
```

Hints

```
/**
 * Print first 20 Fibonacci numbers and their average
 */
public class Fibonacci {
   public static void main (String[] args) {
      int n = 3;          // The index n for F(n), starting from n=3, as n=1 and n=2 are pre-defined
      int fn;             // F(n) to be computed
      int fnMinus1 = 1;   // F(n-1), init to F(2)
      int fnMinus2 = 1;   // F(n-2), init to F(1)
      int nMax = 20;      // maximum n, inclusive
      int sum = fnMinus1 + fnMinus2;  // Need sum to compute average
      double average;

      System.out.println("The first " + nMax + " Fibonacci numbers are:");
      ......

      while (n <= nMax) {  // n starts from 3
            // n = 3, 4, 5, ..., nMax
         // Compute F(n), print it and add to sum
         ......
         // Increment the index n and shift the numbers for the next iteration
         ++n;
         fnMinus2 = fnMinus1;
         fnMinus1 = fn;
      }

      // Compute and display the average (=sum/nMax).
      // Beware that int/int gives int.
      ......
   }
}
```

Try

1. Tribonacci numbers are a sequence of numbers T(n) similar to Fibonacci numbers, except that a number is formed by adding the three previous numbers, i.e., T(n)=T(n-1)+T(n-2)+T(n-3), T(1) =T(2)=1, and T(3)=2. Write a program called Tribonacci to produce the first twenty Tribonacci numbers.

3.7 ExtractDigits (Decision & Loop)

Write a program called **ExtractDigits** to extract each digit from an int, in the reverse order. For example, if the int is 15423, the output shall be "3 2 4 5 1", with a space separating the digits.

Hints

The coding pattern for extracting individual digits from an integer n is:

1. Use (n % 10) to extract the last (least-significant) digit.
2. Use n = n / 10 to drop the last (least-significant) digit.
3. Repeat if (n > 0), i.e., more digits to extract.

Take note that n is destroyed in the process. You may need to clone a copy.

```
int n = ...;
while (n > 0) {
   int digit = n % 10;  // Extract the least-significant digit
   // Print this digit
   ......
   n = n / 10;  // Drop the least-significant digit and repeat the loop
}
```

4. Exercises on Input, Decision and Loop

4.1 Add2Integer (Input)

Write a program called Add2Integers that prompts user to enter two integers. The program shall read the two integers as int; compute their sum; and print the result. For example,

```
Enter first integer: 8
Enter second integer: 9
The sum is: 17
```

Hints

```
import java.util.Scanner;   // For keyboard input
/**
 * 1. Prompt user for 2 integers
 * 2. Read inputs as "int"
 * 3. Compute their sum in "int"
 * 4. Print the result
 */
public class Add2Integers {  // Save as "Add2Integers.java"
   public static void main (String[] args) {
      // Declare variables
      int number1, number2, sum;

      // Put up prompting messages and read inputs as "int"
      Scanner in = new Scanner(System.in);  // Scan the keyboard for input
      System.out.print("Enter first integer: ");  // No newline for prompting message
      number1 = in.nextInt();                     // Read next input as "int"
      ......
      in.close();  // Close Scanner

      // Compute sum
      sum = ......

      // Display result
      System.out.println("The sum is: " + sum);   // Print with newline
   }
}
```

4.2 SumProductMinMax3 (Arithmetic & Min/Max)

Write a program called SumProductMinMax3 that prompts user for three integers. The program shall read the inputs as int; compute the sum, product, minimum and maximum of the three integers; and print the results. For example,

```
Enter 1st integer: 8
Enter 2nd integer: 2
Enter 3rd integer: 9
The sum is: 19
The product is: 144
The min is: 2
The max is: 9
```

Hints

```
    // Declare variables
    int number1, number2, number3;  // The 3 input integers
    int sum, product, min, max;     // To compute these

    // Prompt and read inputs as "int"
    Scanner in = new Scanner(System.in);  // Scan the keyboard
    ......
    ......
    in.close();

    // Compute sum and product
    sum = ......
    product = ......

    // Compute min
    // The "coding pattern" for computing min is:
    // 1. Set min to the first item
    // 2. Compare current min with the second item and update min if second item is smaller
    // 3. Repeat for the next item
    min = number1;        // Assume min is the 1st item
    if (number2 < min) {  // Check if the 2nd item is smaller than current min
      min = number2;      // Update min if so
    }
    if (number3 < min) {  // Continue for the next item
      min = number3;
    }

    // Compute max - similar to min
    ......

    // Print results
    ......
```

Try

1. **Write a program called SumProductMinMax5 that prompts user for five integers. The program shall read the inputs as int; compute the sum, product, minimum and maximum of the five integers; and print the results. Use five int variables: number1, number2, ..., number5 to store the inputs.**

4.3 CircleComputation (double & printf())

Write a program called **CircleComputation** that prompts user for the radius of a circle in floating point number. The program shall read the input as double; compute the diameter, circumference, and area of the circle in double; and print the values rounded to 2 decimal places. Use System-provided constant Math.PI for pi. The formulas are:

```
diameter = 2.0 * radius;
area = Math.PI * radius * radius;
circumference = 2.0 * Math.PI * radius;
```

Hints

```
    // Declare variables
    double radius, diameter, circumference, area;  // inputs and results - all in double
    ......

    // Prompt and read inputs as "double"
```

```
System.out.print("Enter the radius: ");
radius = in.nextDouble();  // read input as double

// Compute in "double"
......

// Print results using printf() with the following format specifiers:
//   %.2f for a double with 2 decimal digits
//   %n for a newline
System.out.printf("Diameter is: %.2f%n", diameter);
......
```

Try

1. Write a program called **SphereComputation** that prompts user for the radius of a sphere in floating point number. The program shall read the input as double; compute the volume and surface area of the sphere in double; and print the values rounded to 2 decimal places. The formulas are:

```
surfaceArea = 4 * Math.PI * radius * radius;
volume = 4 /3 * Math.PI * radius * radius * radius;   // But this does not work in programming?! Why?
```

Take note that you cannot name the variable surface area with a space or surface-area with a dash. Java's naming convention is surfaceArea. Other languages recommend surface_area with an underscore.

2. Write a program called **CylinderComputation** that prompts user for the base radius and height of a cylinder in floating point number. The program shall read the inputs as double; compute the base area, surface area, and volume of the cylinder; and print the values rounded to 2 decimal places. The formulas are:

```
baseArea = Math.PI * radius * radius;
surfaceArea = 2.0 * Math.PI * radius + 2.0 * baseArea;
volume = baseArea * height;
```

4.4 Swap2Integers

Write a program called Swap2Integers that prompts user for two integers. The program shall read the inputs as int, save in two variables called number1 and number2; swap the contents of the two variables; and print the results. For examples,

```
Enter first integer: 9
Enter second integer: -9
After the swap, first integer is: -9, second integer is: 9
```

Hints

To swap the contents of two variables x and y, you need to introduce a temporary storage, say temp, and do: temp ⇐ x; x ⇐ y; y ⇐ temp.

4.5 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is $85000, the income tax payable is $20000*0% + $20000*10% + $20000*20% + $25000*30%.

Write a program called **IncomeTaxCalculator** that reads the taxable income (in int). The program shall calculate the income tax payable (in double); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

Hints

```
// Declare constants first (variables may use these constants)
// The keyword "final" marked these as constant (i.e., cannot be changed).
// Use uppercase words joined with underscore to name constants
final double TAX_RATE_ABOVE_20K = 0.1;
final double TAX_RATE_ABOVE_40K = 0.2;
final double TAX_RATE_ABOVE_60K = 0.3;

// Declare variables
int taxableIncome;
double taxPayable;
......

// Compute tax payable in "double" using a nested-if to handle 4 cases
if (taxableIncome <= 20000) {         // [0, 20000]
  taxPayable = ......;
} else if (taxableIncome <= 40000) {  // [20001, 40000]
  taxPayable = ......;
} else if (taxableIncome <= 60000) {  // [40001, 60000]
  taxPayable = ......;
} else {                              // [60001, ]
  taxPayable = ......;
}
// Alternatively, you could use the following nested-if conditions
// but the above follows the table data
//if (taxableIncome > 60000) {          // [60001, ]
//   ......
//} else if (taxableIncome > 40000) {   // [40001, 60000]
//   ......
//} else if (taxableIncome > 20000) {   // [20001, 40000]
//   ......
//} else {                              // [0, 20000]
//   ......
//}

// Print results rounded to 2 decimal places
System.out.printf("The income tax payable is: $%.2f%n", ...);
```

Try

Suppose that a 10% tax rebate is announced for the income tax payable, capped at $1,000, modify your program to handle the tax rebate. For example, suppose that the tax payable is $12,000, the rebate is $1,000, as 10% of $12,000 exceed the cap.

4.6 IncomeTaxCalculatorWithSentinel (Decision & Loop)

Based on the previous exercise, write a program called IncomeTaxCalculatorWithSentinel which shall repeat the calculation until user enter -1. For example,

```
Enter the taxable income (or -1 to end): $41000
The income tax payable is: $2200.00

Enter the taxable income (or -1 to end): $62000
The income tax payable is: $6600.00

Enter the taxable income (or -1 to end): $73123
The income tax payable is: $9936.90

Enter the taxable income (or -1 to end): $84328
The income tax payable is: $13298.40

Enter the taxable income: $-1
bye!
```

The -1 is known as the sentinel value. (Wiki: In programming, a sentinel value, also referred to as a flag value, trip value, rogue value, signal value, or dummy data, is a special value which uses its presence as a condition of termination.)

Hints

The coding pattern for handling input with sentinel value is as follows:

```
// Declare constants first
final int SENTINEL = -1;    // Terminating value for input
......
// Declare variables
int taxableIncome;
double taxPayable;
......

// Read the first input to "seed" the while loop
System.out.print("Enter the taxable income (or -1 to end): $");
taxableIncome = in.nextInt();

while (taxableIncome != SENTINEL) {
  // Compute tax payable
  ......
  // Print result
  ......

  // Read the next input
  System.out.print("Enter the taxable income (or -1 to end): $");
  taxableIncome = in.nextInt();
    // Repeat the loop body, only if the input is not the SENTINEL value.
    // Take note that you need to repeat these two statements inside/outside the loop!
}
System.out.println("bye!");
```

Take note that we repeat the input statements inside and outside the loop. Repeating statements is NOT a good programming practice. This is because it is easy to repeat (Cntl-C/Cntl-V), but hard to maintain and synchronize the repeated statements. In this case, we have no better choices!

4.7 PensionContributionCalculatorWithSentinel (Decision & Loop)

Based on the previous PensionContributionCalculator,
write a program called **PensionContributionCalculatorWithSentinel** which shall repeat the calculations until user enter -1 for the salary. For examples,

```
Enter the monthly salary (or -1 to end): $5123
Enter the age: 21
The employee's contribution is: $1024.60
The employer's contribution is: $870.91
The total contribution is: $1895.51

Enter the monthly salary (or -1 to end): $5123
Enter the age: 64
The employee's contribution is: $384.22
The employer's contribution is: $461.07
The total contribution is: $845.30

Enter the monthly salary (or -1 to end): $-1
bye!
```

Hints

```
// Read the first input to "seed" the while loop
System.out.print("Enter the monthly salary (or -1 to end): $");
salary = in.nextInt();

while (salary != SENTINEL) {
  // Read the remaining
  System.out.print("Enter the age: ");
  age = in.nextInt();

  ......
  ......

  // Read the next input and repeat
  System.out.print("Enter the monthly salary (or -1 to end): $");
  salary = in.nextInt();
}
```

4.8 SalesTaxCalculator (Decision & Loop)

A sales tax of 7% is levied on all goods and services consumed. It is also mandatory that all the price tags should include the sales tax. For example, if an item has a price tag of $107, the actual price is $100 and $7 goes to the sales tax.
Write a program using a loop to continuously input the tax-inclusive price (in double); compute the actual price and the sales tax (in double); and print the results rounded to 2 decimal places. The program shall terminate in response to input of -1; and print the total price, total actual price, and total sales tax. For examples,

```
Enter the tax-inclusive price in dollars (or -1 to end): 107
Actual Price is: $100.00, Sales Tax is: $7.00

Enter the tax-inclusive price in dollars (or -1 to end): 214
Actual Price is: $200.00, Sales Tax is: $14.00

Enter the tax-inclusive price in dollars (or -1 to end): 321
Actual Price is: $300.00, Sales Tax is: $21.00

Enter the tax-inclusive price in dollars (or -1 to end): -1
Total Price is: $642.00
Total Actual Price is: $600.00
Total Sales Tax is: $42.00
```

Hints

```
// Declare constants
final double SALES_TAX_RATE = 0.07;
final int SENTINEL = -1;        // Terminating value for input

// Declare variables
double price, actualPrice, salesTax;  // inputs and results
```

```
   double totalPrice = 0.0, totalActualPrice = 0.0, totalSalesTax = 0.0;  // to accumulate
   ......

   // Read the first input to "seed" the while loop
   System.out.print("Enter the tax-inclusive price in dollars (or -1 to end): ");
   price =  in.nextDouble();

   while (price != SENTINEL) {
     // Compute the tax
     ......
     // Accumulate into the totals
     ......
     // Print results
     ......

     // Read the next input and repeat
     System.out.print("Enter the tax-inclusive price in dollars (or -1 to end): ");
     price =  in.nextDouble();
   }
   // print totals
   ......
```

4.9 ReverseInt (Loop with Modulus/Divide)

Write a program that prompts user for a positive integer. The program shall read the input as int; and print the "reverse" of the input integer. For examples,

```
Enter a positive integer: 12345
The reverse is: 54321
```

Hints

Use the following coding pattern which uses a while-loop with repeated modulus/divide operations to extract and drop the last digit of a positive integer.

```
   // Declare variables
   int inNumber;   // to be input
   int inDigit;    // each digit
   ......

   // Extract and drop the "last" digit repeatably using a while-loop with modulus/divide operations
   while (inNumber > 0) {
     inDigit = inNumber % 10; // extract the "last" digit
     // Print this digit (which is extracted in reverse order)
     ......
     inNumber /= 10;          // drop "last" digit and repeat
   }
   ......
```

4.10 SumOfDigitsInt (Loop with Modulus/Divide)

Write a program that prompts user for a positive integer. The program shall read the input as int; compute and print the sum of all its digits. For examples,

```
Enter a positive integer: 12345
The sum of all digits is: 15
```

Hints

See "ReverseInt".

4.11 InputValidation (Loop with boolean flag)

Your program often needs to validate the user's inputs, e.g., marks shall be between 0 and 100.

Write a program that prompts user for an integer between 0-10 or 90-100. The program shall read the input as int; and repeat until the user enters a valid input. For examples,

```
Enter a number between 0-10 or 90-100: -1
Invalid input, try again...
Enter a number between 0-10 or 90-100: 50
Invalid input, try again...
Enter a number between 0-10 or 90-100: 101
Invalid input, try again...
Enter a number between 0-10 or 90-100: 95
You have entered: 95
```

Hints

Use the following coding pattern which uses a do-while loop controlled by a boolean flag to do input validation. We use a do-while instead of while-do loop as we need to execute the body to prompt and process the input at least once.

```
// Declare variables
int numberIn;      // to be input
boolean isValid;   // boolean flag to control the loop
......

// Use a do-while loop controlled by a boolean flag
// to repeatably read the input until a valid input is entered
isValid = false;   // default assuming input is not valid
do {
  // Prompt and read input
  ......

  // Validate input by setting the boolean flag accordingly
  if (numberIn ......) {
    isValid = true;   // exit the loop
  } else {
    System.out.println(......);  // Print error message and repeat
  }
} while (!isValid);
......
```

4.12 AverageWithInputValidation (Loop with boolean flag)

Write a program that prompts user for the mark (between 0-100 in int) of 3 students; computes the average (in double); and prints the result rounded to 2 decimal places. Your program needs to perform input validation. For examples,

```
Enter the mark (0-100) for student 1: 56
Enter the mark (0-100) for student 2: 101
Invalid input, try again...
Enter the mark (0-100) for student 2: -1
Invalid input, try again...
Enter the mark (0-100) for student 2: 99
Enter the mark (0-100) for student 3: 45
The average is: 66.67
```

Hints

```
// Declare constant
```

```
final int NUM_STUDENTS = 3;

// Declare variables
int numberIn;
boolean isValid;   // boolean flag to control the input validation loop
int sum = 0;
double average;
......

for (int studentNo = 1; studentNo <= NUM_STUDENTS; ++studentNo) {
   // Prompt user for mark with input validation
   ......
   isValid = false;   // reset assuming input is not valid
   do {
      ......
   } while (!isValid);

   sum += ......;
}
......
```

5. Exercises on Nested-Loops

5.1 **SquarePattern** (nested-loop)

Write a program called **SquarePattern** that prompts user for the size (a non-negative integer in int); and prints the following square pattern using two nested for-loops.

```
Enter the size: 5
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Hints

The code pattern for printing 2D patterns using nested loops is:

```
// Outer loop to print each of the rows
for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
   // Inner loop to print each of the columns of a particular row
   for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
      System.out.print( ...... );   // Use print() without newline inside the inner loop
      ......
   }
   // Print a newline after printing all the columns
   System.out.println();
}
```

Notes

1. You should name the loop indexes row and col, NOT i and j, or x and y, or a and b, which are meaningless.
2. The row and col could start at 1 (and upto size), or start at 0 (and upto size-1). As computer counts from 0, it is probably more efficient to start from 0. However, since humans counts from 1, it is easier to read if you start from 1.

Try

Rewrite the above program using nested while-do loops.

5.2 CheckerPattern (nested-loop)

Write a program called **CheckerPattern** that prompts user for the size (a non-negative integer in int); and prints the following checkerboard pattern.

```
Enter the size: 7
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

Hints

```
// Outer loop to print each of the rows
    for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
       // Inner loop to print each of the columns of a particular row
       for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
          if ((row % 2) == 0) {   // row 2, 4, 6, ...
             ......
          }
          System.out.print( ...... );   // Use print() without newline inside the inner loop
          ......
       }
       // Print a newline after printing all the columns
       System.out.println();
    }
```

5.3 TimeTable (nested-loop)

Write a program called **TimeTable** that prompts user for the size (a positive integer in int); and prints the multiplication table as shown:

```
Enter the size: 10
 * |   1   2   3   4   5   6   7   8   9  10
--------------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   3   6   9  12  15  18  21  24  27  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

Hints

1. Use printf() to format the output, e.g., each cell is %4d.
2. See "Java Basics" article.

5.4 TriangularPattern (nested-loop)

Write 4 programs called **TriangularPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints each of the patterns as shown:

```
Enter the size: 8
```

```
#                  # # # # # # # #    # # # # # # # #                  #
# #                # # # # # # #        # # # # # # #                # #
# # #              # # # # # #            # # # # # #              # # #
# # # #            # # # # #                # # # # #            # # # #
# # # # #          # # # #                    # # # #          # # # # #
# # # # # #        # # #                        # # #        # # # # # #
# # # # # # #      # #                            # #      # # # # # # #
# # # # # # # #    #                                #    # # # # # # # #
   (a)                (b)                 (c)               (d)
```

Hints

1. On the main diagonal, row = col. On the opposite diagonal, row + col = size + 1, where row and col begin from 1.
2. You need to print the leading blanks, in order to push the # to the right. The trailing blanks are optional, which does not affect the pattern.
3. For pattern (a), if (row >= col) print #. Trailing blanks are optional.
4. For pattern (b), if (row + col <= size + 1) print #. Trailing blanks are optional.
5. For pattern (c), if (row >= col) print #; else print blank. Need to print the leading blanks.
6. For pattern (d), if (row + col >= size + 1) print #; else print blank. Need to print the leading blanks.
7. The coding pattern is:

```
// Outer loop to print each of the rows
   for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
      // Inner loop to print each of the columns of a particular row
      for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
         if (......) {
            System.out.print("# ");
         } else {
            System.out.print("  ");  // Need to print the "leading" blanks
         }
      }
      // Print a newline after printing all the columns
      System.out.println();
   }
```

5.5 BoxPattern (nested-loop)

Write 4 programs called **BoxPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints the pattern as shown:

```
Enter the size: 8

# # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #
#        #    #                 #     #     #     # #    # #
#        #     #              #        #  #      #  #  #  #
#        #       #          #           #       #   #   #
#        #        #        #           #  #      #  #  #  #
#        #          #     #           #     #     # #    # #
# # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #
   (a)            (b)            (c)            (d)            (e)
```

Hints

1. On the main diagonal, row = col. On the opposite diagonal, row + col = size + 1, where row and col begin from 1.
2. For pattern (a), if (row == 1 || row == size || col == 1 || col == size) print #; else print blank. Need to print the intermediate blanks.
3. For pattern (b), if (row == 1 || row == size || row == col) print #; else print blank.

5.6 HillPattern (nested-loop)

Write 3 programs called **HillPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints the pattern as shown:

```
Enter the rows: 6

          #             # # # # # # # # # # #             #             # # # # # # # # # # #
        # # #             # # # # # # # # #             # # #           # # # # #   # # # # #
      # # # # #             # # # # # # #             # # # # #         # # # #       # # # #
    # # # # # # #             # # # # #             # # # # # # #       # # #           # # #
  # # # # # # # # #             # # #             # # # # # # # # #     # #               # #
# # # # # # # # # # #             #             # # # # # # # # # # #   #                   #
         (a)                     (b)              # # # # # # # # #     # #               # #
                                                    # # # # # # #       # # #           # # #
                                                      # # # # #         # # # #       # # # #
                                                        # # #           # # # # #   # # # # #
                                                          #             # # # # # # # # # # #
                                                         (c)                     (d)
```

Hints

For pattern (a):

```
for (int row = 1; ......) {
   // numCol = 2*numRows - 1
   for (int col = 1; ......) {
      if ((row + col >= numRows + 1) && (row >= col - numRows + 1)) {
         ......;
      } else {
         ......;
      }
   }
   ......;
}
```

or, use 2 sequential inner loops to print the columns:

```
for (int row = 1; row <= rows; row++) {
   for (int col = 1; col <= rows; col++) {
      if ((row + col >= rows + 1)) {
         ......
      } else {
         ......
      }
   }
   for (int col = 2; col <= rows; col++) {  // skip col = 1
      if (row >= col) {
         ......
      } else {
         ......
      }
   }
   ......
}
```

5.7 NumberPattern (nested-loop)

Write 4 programs called **NumberPatternX** (X = A, B, C, D) that prompts user for the size (a non-negative integer in int); and prints the pattern as shown:

```
Enter the size: 8

1                 1 2 3 4 5 6 7 8                 1      8 7 6 5 4 3 2 1
1 2                 1 2 3 4 5 6 7               2 1       7 6 5 4 3 2 1
1 2 3                 1 2 3 4 5 6             3 2 1        6 5 4 3 2 1
1 2 3 4                 1 2 3 4 5           4 3 2 1         5 4 3 2 1
1 2 3 4 5                 1 2 3 4         5 4 3 2 1          4 3 2 1
1 2 3 4 5 6                 1 2 3       6 5 4 3 2 1           3 2 1
1 2 3 4 5 6 7                 1 2     7 6 5 4 3 2 1            2 1
1 2 3 4 5 6 7 8                 1   8 7 6 5 4 3 2 1             1
    (a)                (b)                 (c)               (d)
```

6. Magic(Special) Numbers

6.1. Amicable umbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Amicalble Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368). For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

```
220 and 284 are Amicable Numbers.

Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110

1+2+4+5+10+11+20+22+44+55+110 = 284

Divisors of 284 = 1, 2, 4, 71, 142
1+2+4+71+142 = 220
```

6.2. Armstrong Number

Armstrong number is a positive number if it is equal to the sum of cubes of its digits is called Armstrong number and if its sum is not equal to the number then it's not a Armstrong number. For example,

```
Enter number=145
145 is not an Armstrong Number

Enter number = 153
153 is an Armstrong Number
```

Hints

```
Examples: 153 is Armstrong

(1*1*1)+(5*5*5)+(3*3*3) = 153
```

6.3. Capricorn Number

A number is called Capricorn or Kaprekar number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number : 45
45 is a Capricorn/Kaprekar number
Enter a number : 297
297 is a Capricorn/Kaprekar number
Enter a number : 44
44 is not a Capricorn/Kaprekar number
```

Hints

```
Number = 45
(45)2   = 2025

All parts for 2025:
202     + 5    =     207 (not 45)
20       + 25  =     45
2+  025     =    27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a Java program to generate and show all Kaprekar numbers less than 1000.

6.4. Circular Prime

A circular prime is a prime number with the property that the number generated at each intermediate step when cyclically permuting its digits will be prime. For example, 1193 is a circular prime, since 1931, 9311 and 3119 all are also prime. For example,

```
Enter a number : 137
137 is a Circular Prime
Enter a number : 44
44 is not a Circular Prime
```

6.5. Happy Number

A happy number is a natural number in a given number base that eventually reaches 1 when iterated over the perfect digital invariant function for. Those numbers that do not end in 1 are -unhappy numbers. For example,

```
Enter a number : 31
31 is a Happy number

Enter a number : 32
32 is not a Happy number
```

6.6. Automorphic Number

An Automorphic number is a number whose square “ends” in the same digits as the number itself. For example,

```
Enter a number : 5
5 is a Automorphic Number

Enter a number : 25
25 is a Automorphic Number

Enter a number : 2
2 is not a Automorphic Number
```

Hints

```
5*5 = 25, 6*6 = 36, 25*25 = 625

5,6,25 are automorphic numbers
```

6.7. Disarium Number

A number is called Disarium number if the sum of its power of the positions from left to right is equal to the number. For example,

```
Enter a number : 135
135 is a Disarium Number

Enter a number : 32
32 is not a Disarium Number
```

Hints

$1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135$

6.8. Magic Number

Magic number is the if the sum of its digits recursively are calculated till a single digit If the single digit is 1 then the number is a magic number. Magic number is very similar with Happy Number. For example,

```
Enter a number : 226
226 is a Magic Number

Enter a number : 32
32 is not a Magic Number
Enter number = 541
153 is a Magic Number
```

Hints

226 is said to be a magic number

2+2+6=10 sum of digits is 10 then again 1+0=1 now we get a single digit number is 1.if we single digit number will now 1 them it would not a magic number.

6.9. Neon Number

A neon number is a number where the sum of digits of square of the number is equal to the number. For example if the input number is 9, its square is 9*9 = 81 and sum of the digits is 9. i.e. 9 is a neon number. For example,

```
Enter a number: 9
9 is a Neon Number

Enter a number: 8
8 is not a Neon Number
```

6.10. Palindromic Number

A palindromic number is a number that remains the same when its digits are reversed. For example,

```
Enter a number : 16461
16461 is a Palendromic Number

Enter a number : 1234
1234 is not a Plaindromic Number
```

6.11. Perfect Number

A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For instance, 6 has divisors 1, 2 and 3, and 1 + 2 + 3 = 6, so 6 is a perfect number. For example,

```
Enter a number : 6
6 is a Perfect Number
```

```
Enter a number : 3
3 is not a Perfect Number
```

6.12. Special Number

A number is said to be special number when the sum of factorial of its digits is equal to the number itself. Example- 145 is a Special Number as 1!+4!+5!=145. For example,

```
Enter a number : 145
145 is a Special Number

Enter a number : 23
23 is not a Special Number
```

6.13. Spy Number

A spy number is a number where the sum of its digits equals the product of its digits. For example, 1124 is a spy number, the sum of its digits is 1+1+2+4=8 and the product of its digits is 1*1*2*4=8. For example,

```
Enter a number : 1124
1124 is a Spy Number

Enter a number : 12
12 is not a Spy Number
```

6.14. Ugly Number

A number is said to be an Ugly number if positive numbers whose prime factors only include 2, 3, 5. For example, 6(2×3), 8(2x2x2), 15(3×5) are ugly numbers while 14(2×7) is not ugly since it includes another prime factor 7. Note that 1 is typically treated as an ugly number. For example,

```
Enter a number : 6
6 is an Ugly Number

Enter a number : 14
14 is not an Ugly Number
```

7. Exercises on String and char Operations

7.1 ReverseString (String & char)

Write a program called **ReverseString**, which prompts user for a String, and prints the reverse of the String by extracting and processing each character. The output shall look like:

```
Enter a String: abcdef
The reverse of the String "abcdef" is "fedcba".
```

Hints

For a String called inStr, you can use inStr.length() to get the length of the String; and inStr.charAt(idx) to retrieve the char at the idx position, where idx begins at 0, up to instr.length() - 1.

```
// Define variables
String inStr;       // input String
int inStrLen;       // length of the input String
......

// Prompt and read input as "String"
System.out.print("Enter a String: ");
inStr = in.next();   // use next() to read a String
inStrLen = inStr.length();

// Use inStr.charAt(index) in a loop to extract each character
```

```
   // The String's index begins at 0 from the left.
   // Process the String from the right
   for (int charIdx = inStrLen - 1; charIdx >= 0; --charIdx) {
      // charIdx = inStrLen-1, inStrLen-2, ... ,0
     ......
   }
```

7.2 CountVowelsDigits (String & char)

Write a program called **CountVowelsDigits**, which prompts the user for a String, counts the number of vowels (a, e, i, o, u, A, E, I, O, U) and digits (0-9) contained in the string, and prints the counts and the percentages (rounded to 2 decimal places). For example,

```
Enter a String: testing12345
Number of vowels: 2 (16.67%)
Number of digits: 5 (41.67%)
```

Hints

1. To check if a char c is a digit, you can use boolean expression (c >= '0' && c <= '9'); or use built-in boolean function Character.isDigit(c).
2. You could use in.next().toLowerCase() to convert the input String to lowercase to reduce the number of cases.
3. To print a % using printf(), you need to use %%. This is because % is a prefix for format specifier in printf(), e.g., %d and %f.

7.3 PhoneKeyPad (String & char)

On you phone keypad, the alphabets are mapped to digits as follows:

ABC(2), DEF(3), GHI(4), JKL(5), MNO(6), PQRS(7), TUV(8), WXYZ(9).

Write a program called **PhoneKeyPad**, which prompts user for a String (case insensitive), and converts to a sequence of keypad digits. Use (a) a nested-if, (b) a switch-case-default.

Hints

1. You can use in.next().toLowerCase() to read a String and convert it to lowercase to reduce your cases.
2. In switch-case, you can handle multiple cases by omitting the break statement, e.g.,

```
switch (inChar) {
  case 'a': case 'b': case 'c':  // No break for 'a' and 'b', fall thru 'c'
    System.out.print(2); break;
  case 'd': case 'e': case 'f':
    ......
  default:
    ......
}
```

7.4 Caesar's Code (String & char)

Caesar's Code is one of the simplest encryption techniques. Each letter in the plaintext is replaced by a letter some fixed number of position (n) down the alphabet cyclically. In this exercise, we shall pick n=3. That is, 'A' is replaced by 'D', 'B' by 'E', 'C' by 'F', ..., 'X' by 'A', ..., 'Z' by 'C'.

Write a program called **CaesarCode** to cipher the Caesar's code. The program shall prompt user for a plaintext string consisting of mix-case letters only; compute the ciphertext; and print the ciphertext in uppercase. For example,

```
Enter a plaintext string: Testing
```

```
The ciphertext string is: WHVWLQJ
```

Hints

1. Use in.next().toUpperCase() to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases ('A'-'Z'). But it is much better to consider 'A' to 'W' as one case; 'X', 'Y' and 'Z' as 3 separate cases.
3. Take note that char 'A' is represented as Unicode number 65 and char 'D' as 68. However, 'A' + 3 gives 68. This is because char + int is implicitly casted to int + int which returns an int value. To obtain a char value, you need to perform explicit type casting using (char)('A' + 3). Try printing ('A' + 3) with and without type casting.

7.5 Decipher Caesar's Code (String & char)

Write a program called **DecipherCaesarCode** to decipher the Caesar's code described in the previous exercise. The program shall prompts user for a ciphertext string consisting of mix-case letters only; compute the plaintext; and print the plaintext in uppercase. For example,

```
Enter a ciphertext string: wHVwLQJ
The plaintext string is: TESTING
```

7.6 Exchange Cipher (String & char)

This simple cipher exchanges 'A' and 'Z', 'B' and 'Y', 'C' and 'X', and so on.
Write a program called **ExchangeCipher** that prompts user for a plaintext string consisting of mix-case letters only. You program shall compute the ciphertext; and print the ciphertext in uppercase. For examples,

```
Enter a plaintext string: abcXYZ
The ciphertext string is: ZYXCBA
```

Hints

1. Use in.next().toUpperCase() to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases ('A'-'Z'), or use the following relationship:

```
'A' + 'Z' == 'B' + 'Y' == 'C' + 'X' == ... == plainTextChar + cipherTextChar
Hence, cipherTextChar = 'A' + 'Z' - plainTextChar
```

7.7 TestPalindromicWord and TestPalindromicPhrase (String & char)

A word that reads the same backward as forward is called a palindrome, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive).

Write a program called **TestPalindromicWord**, that prompts user for a word and prints ""xxx" is|is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization).

Modify your program (called **TestPalindromicPhrase**) to check for palindromic phrase. Use in.nextLine() to read a line of input.

Hints

1. **Maintain two indexes, forwardIndex (fIdx) and backwardIndex (bIdx), to scan the phrase forward and backward.**

```
int fIdx = 0, bIdx = strLen - 1;
while (fIdx < bIdx) {
  ......
  ++fIdx;
```

```
         --bIdx;
       }
       // or
       for (int fIdx = 0, bIdx = strLen - 1; fIdx < bIdx; ++fIdx, --bIdx) {
         ......
       }
```

2. You can check if a char c is a letter either using built-in boolean function Character.isLetter(c); or boolean expression (c >= 'a' && c <= 'z'). Skip the index if it does not contain a letter.

7.8 CheckBinStr (String & char)

The binary number system uses 2 symbols, 0 and 1. Write a program called **CheckBinStr** to verify a binary string. The program shall prompt user for a binary string; and decide if the input string is a valid binary string. For example,

```
Enter a binary string: 10101100
"10101100" is a binary string

Enter a binary string: 10120000
"10120000" is NOT a binary string
```

Hints

Use the following coding pattern which involves a boolean flag to check the input string.

```
// Declare variables
  String inStr;     // The input string
  int inStrLen;     // The length of the input string
  char inChar;      // Each char of the input string
  boolean isValid;  // "is" or "is not" a valid binary string?
  ......

  isValid = true;  // Assume that the input is valid, unless our check fails
  for (......) {
    inChar = ......;
    if (!(inChar == '0' || inChar == '1')) {
      isValid = false;
      break;  // break the loop upon first error, no need to continue for more errors
              // If this is not encountered, isValid remains true after the loop.
    }
  }
  if (isValid) {
    System.out.println(......);
  } else {
    System.out.println(......);
  }
  // or using one liner
  //System.out.println(isValid ? ... : ...);
```

7.9 CheckHexStr (String & char)

The hexadecimal (hex) number system uses 16 symbols, 0-9 and A-F (or a-f). Write a program to verify a hex string. The program shall prompt user for a hex string; and decide if the input string is a valid hex string. For examples,

```
Enter a hex string: 123aBc
"123aBc" is a hex string

Enter a hex string: 123aBcx
"123aBcx" is NOT a hex string
```

Hints

```
if (!((inChar >= '0' && inChar <= '9')
      || (inChar >= 'A' && inChar <= 'F')
      || (inChar >= 'a' && inChar <= 'f'))) {  // Use positive logic and then reverse
   ......
}
```

7.10 Bin2Dec (String & char)

Write a program called **Bin2Dec** to convert an input binary string into its equivalent decimal number. Your output shall look like:

```
Enter a Binary string: 1011
The equivalent decimal number for binary "1011" is: 11

Enter a Binary string: 1234
error: invalid binary string "1234"
```

Hints

See "Code Example".

7.11 Hex2Dec (String & char)

Write a program called **Hex2Dec** to convert an input hexadecimal string into its equivalent decimal number. Your output shall look like:

```
Enter a Hexadecimal string: 1a
The equivalent decimal number for hexadecimal "1a" is: 26

Enter a Hexadecimal string: 1y3
error: invalid hexadecimal string "1y3"
```

Hints

See "Code Example".

7.12 Oct2Dec (String & char)

Write a program called **Oct2Dec** to convert an input Octal string into its equivalent decimal number. For example,

```
Enter an Octal string: 147
The equivalent decimal number "147" is: 103
```

8. Exercises on Arrays

8.1 PrintArray (Array)

Write a program called PrintArray which prompts user for the number of items in an array (a non-negative integer), and saves it in an int variable called NUM_ITEMS. It then prompts user for the values of all the items and saves them in an int array called items. The program shall then print the contents of the array in the form of [x1, x2, ..., xn]. For example,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 3 2 5 6 9
The values are: [3, 2, 5, 6, 9]
```

Hints

```
      // Declare variables
      tinal int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......
```

```
// Prompt for for the number of items and read the input as "int"
......
NUM_ITEMS = ......

// Allocate the array
items = new int[NUM_ITEMS];

// Prompt and read the items into the "int" array, if array length > 0
if (items.length > 0) {
  ......
  for (int i = 0; i < items.length; ++i) {  // Read all items
    ......
  }
}

// Print array contents, need to handle first item and subsequent items differently
......
for (int i = 0; i < items.length; ++i) {
  if (i == 0) {
    // Print the first item without a leading commas
    ......
  } else {
    // Print the subsequent items with a leading commas
    ......
  }
  // or, using a one liner
  //System.out.print((i == 0) ? ...... : ......);
}
```

8.2 PrintArrayInStars (Array)

Write a program called **printArrayInStars** which prompts user for the number of items in an array (a non-negative integer), and saves it in an int variable called NUM_ITEMS. It then prompts user for the values of all the items (non-negative integers) and saves them in an int array called items. The program shall then print the contents of the array in a graphical form, with the array index and values represented by number of stars. For examples,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 7 4 3 0 7
0: *******(7)
1: ****(4)
2: ***(3)
3: (0)
4: *******(7)
```

Hints

```
// Declare variables
final int NUM_ITEMS;
int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
......
......
// Print array in "index: number of stars" using a nested-loop
// Take note that rows are the array indexes and columns are the value in that index
for (int idx = 0; idx < items.length; ++idx) {  // row
  System.out.print(idx + ": ");
  // Print value as the number of stars
  for (int starNo = 1; starNo <= items[idx]; ++starNo) {  // column
    System.out.print("*");
  }
  ......
```

```
    }
    ......
```

8.3 GradesStatistics (Array)

Write a program which prompts user for the number of students in a class (a non-negative integer), and saves it in an int variable called numStudents. It then prompts user for the grade of each of the students (integer between 0 to 100) and saves them in an int array called grades. The program shall then compute and print the average (in double rounded to 2 decimal places) and minimum/maximum (in int).

```
Enter the number of students: 5
Enter the grade for student 1: 98
Enter the grade for student 2: 78
Enter the grade for student 3: 78
Enter the grade for student 4: 87
Enter the grade for student 5: 76
The average is: 83.40
The minimum is: 76
The maximum is: 98
```

8.4 Hex2Bin (Array for Table Lookup)

Write a program called **Hex2Bin** that prompts user for a hexadecimal string and print its equivalent binary string. The output shall look like:

```
Enter a Hexadecimal string: 1abc
The equivalent binary for hexadecimal "1abc" is: 0001 1010 1011 1100
```

Hints

1. Use an array of 16 Strings containing binary strings corresponding to hexadecimal number 0-9A-F (or a-f), as follows

```
final String[] HEX_BITS = {"0000", "0001", "0010", "0011",
              "0100", "0101", "0110", "0111",
              "1000", "1001", "1010", "1011",
              "1100", "1101", "1110", "1111"};
```

8.5 Dec2Hex (Array for Table Lookup)

Write a program called **Dec2Hex** that prompts user for a positive decimal number, read as int, and print its equivalent hexadecimal string. The output shall look like:

```
Enter a decimal number: 1234
The equivalent hexadecimal number is 4D2
```

9. Exercises on Methods

9.1 exponent() (method)

Write a method called exponent(int base, int exp) that returns an int value of base raises to the power of exp. The signature of the method is:

```
 public static int exponent(int base, int exp);
```

Assume that exp is a non-negative integer and base is an integer. Do not use any Math library functions.
Also write the main() method that prompts user for the base and exp; and prints the result. For example,

```
Enter the base: 3
Enter the exponent: 4
3 raises to the power of 4 is: 81
```

Hints

```
......
public class Exponent {
  public static void main(String[] args) {
    // Declare variables
    int exp;    // exponent (non-negative integer)
    int base;   // base (integer)
    ......
    // Prompt and read exponent and base
    ......
    // Print result
    System.out.println(base + " raises to the power of " + exp + " is: " + exponent(base, exp));
  }

  // Returns "base" raised to the power "exp"
  public static int exponent(int base, int exp) {
    int product = 1;   // resulting product

    // Multiply product and base for exp number of times
    for (......) {
      product *= base;
    }

    return product;
  }
}
```

9.2 isOdd() (method)

Write a boolean method called isOdd() in a class called **OddEvenTest**, which takes an int as input and returns true if the it is odd. The signature of the method is as follows:

```
public static boolean isOdd(int number);
```

Also write the main() method that prompts user for a number, and prints "ODD" or "EVEN". You should test for negative input. For examples,

```
Enter a number: 9
9 is an odd number

Enter a number: 8
8 is an even number

Enter a number: -5
-5 is an odd number
```

Hints

See Notes.

9.3 hasEight() (method)

Write a boolean method called hasEight(), which takes an int as input and returns true if the number contains the digit 8 (e.g., 18, 168, 1288). The signature of the method is as follows:

```
public static boolean hasEight(int number);
```

Write a program called **MagicSum**, which prompts user for integers (or -1 to end), and produce the sum of numbers containing the digit 8. Your program should use the above methods. A sample output of the program is as follows:

```
Enter a positive integer (or -1 to end): 1
Enter a positive integer (or -1 to end): 2
Enter a positive integer (or -1 to end): 3
Enter a positive integer (or -1 to end): 8
Enter a positive integer (or -1 to end): 88
Enter a positive integer (or -1 to end): -1
The magic sum is: 96
```

Hints

The coding pattern to repeat until input is -1 (called sentinel value) is:

```
final int SENTINEL = -1;  // Terminating input
int number;

// Read first input to "seed" the while loop
System.out.print("Enter a positive integer (or -1 to end): ");
number = in.nextInt();

while (number != SENTINEL) {  // Repeat until input is -1
   ......
   ......

   // Read next input. Repeat if the input is not the SENTINEL
   // Take note that you need to repeat these codes!
   System.out.print("Enter a positive integer (or -1 to end): ");
   number = in.nextInt();
}
```

You can either repeatably use modulus/divide (n%10 and n=n/10) to extract and drop each digit in int; or convert the int to String and use the String's charAt() to inspect each char.

9.4 print() (Array & Method)

Write a method called **print()**, which takes an int array and print its contents in the form of [a1, a2, ..., an]. Take note that there is no comma after the last element. The method's signature is as follows:

```
public static void print(int[] array);
```

Also write a test driver to test this method (you should test on empty array, one-element array, and n-element array).

How to handle double[] or float[]? You need to write a overloaded version for double[] and a overloaded version for float[], with the following signatures:

```
public static void print(double[] array)
public static void print(float[] array)
```

The above is known as method overloading, where the same method name can have many versions, differentiated by its parameter list.

Hints

For the first element, print its value; for subsequent elements, print commas followed by the value.

9.5 arrayToString() (Array & Method)

Write a method called **arrayToString()**, which takes an int array and return a String in the form of [a1, a2, ..., an]. Take note that this method returns a String, the previous exercise returns void but prints the output. The method's signature is as follows:

```
public static String arrayToString(int[] array);
```

Also write a test driver to test this method (you should test on empty array, one-element array, and n-element array).

Notes: This is similar to the built-in function Arrays.toString(). You could study its source code.

9.6 contains() (Array & Method)

Write a boolean method called **contains()**, which takes an array of int and an int; and returns true if the array contains the given int. The method's signature is as follows:

```
public static boolean contains(int[] array, int key);
```

Also write a test driver to test this method.

9.7 search() (Array & Method)

Write a method called **search()**, which takes an array of int and an int; and returns the array index if the array contains the given int; or -1 otherwise. The method's signature is as follows:

```
public static int search(int[] array, int key);
```

Also write a test driver to test this method.

9.8 equals() (Array & Method)

Write a boolean method called **equals()**, which takes two arrays of int and returns true if the two arrays are exactly the same (i.e., same length and same contents). The method's signature is as follows:

```
public static boolean equals(int[] array1, int[] array2)
```

Also write a test driver to test this method.

9.9 copyOf() (Array & Method)

Write a boolean method called **copyOf()**, which takes an int Array and returns a copy of the given array. The method's signature is as follows:

```
public static int[] copyOf(int[] array)
```

Also write a test driver to test this method.

Write another version for **copyOf()** which takes a second parameter to specify the length of the new array. You should truncate or pad with zero so that the new array has the required length.

```
public static int[] copyOf(int[] array, int newLength)
```

NOTES: This is similar to the built-in function Arrays.copyOf().

9.10 swap() (Array & Method)

Write a method called **swap()**, which takes two arrays of int and swap their contents if they have the same length. It shall return true if the contents are successfully swapped. The method's signature is as follows:

```
public static boolean swap(int[] array1, int[] array2)
```

Also write a test driver to test this method.

Hints

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

9.11 reverse() (Array & Method)

Write a method called **reverse()**, which takes an array of int and reverse its contents. For example, the reverse of [1,2,3,4] is [4,3,2,1]. The method's signature is as follows:

```
public static void reverse(int[] array);
```

Take note that the array passed into the method can be modified by the method (this is called "pass by reference"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "pass by value").

Also write a test driver to test this method.

Hints

You might use two indexes in the loop, one moving forward and one moving backward to point to the two elements to be swapped.

```
for (int fIdx = 0, bIdx = array.length - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   // Swap array[fIdx] and array[bIdx]
   // Only need to transverse half of the array elements
}
```

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

9.12 GradesStatistics (Array & Method)

Write a program called **GradesStatistics**, which reads in n grades (of int between 0 and 100, inclusive) and displays the average, minimum, maximum, median and standard deviation. Display the floating-point values upto 2 decimal places. Your output shall look like:

```
Enter the number of students: 4
Enter the grade for student 1: 50
Enter the grade for student 2: 51
Enter the grade for student 3: 56
Enter the grade for student 4: 53
The grades are: [50, 51, 56, 53]
The average is: 52.50
The median is: 52.00
The minimum is: 50
The maximum is: 56
The standard deviation is: 2.29
```

The formula for calculating standard deviation is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i{}^2 - \mu^2}, \text{ where } \mu \text{ is the mean}$$

Hints:

```
public class GradesStatistics {
   public static int[] grades;  // Declare an int[], to be allocated later.
                                // This array is accessible by all the methods.

   public static void main(String[] args) {
      readGrades();  // Read and save the inputs in global int[] grades
      System.out.println("The grades are: ");
      print(grades);
      System.out.println("The average is " + average(grades));
      System.out.println("The median is " + median(grades));
      System.out.println("The minimum is " + min(grades));
      System.out.println("The maximum is " + max(grades));
      System.out.println("The standard deviation is " + stdDev(grades));
   }

   // Prompt user for the number of students and allocate the global "grades" array.
   // Then, prompt user for grade, check for valid grade, and store in "grades".
   public static void readGrades() { ....... }

   // Print the given int array in the form of [x1, x2, x3,..., xn].
   public static void print(int[] array) { ....... }

   // Return the average value of the given int[]
   public static double average(int[] array) { ...... }

   // Return the median value of the given int[]
   // Median is the center element for odd-number array,
   // or average of the two center elements for even-number array.
   // Use Arrays.sort(anArray) to sort anArray in place.
   public static double median(int[] array) { ...... }

   // Return the maximum value of the given int[]
   public static int max(int[] array) {
      int max = array[0];   // Assume that max is the first element
      // From second element, if the element is more than max, set the max to this element.
      ......
   }

   // Return the minimum value of the given int[]
   public static int min(int[] array) { ....... }

   // Return the standard deviation of the given int[]
   public static double stdDev(int[] array) { ....... }
}
```

Take note that besides readGrade() that relies on global variable grades, all the methods are self-contained general utilities that operate on any given array.

9.13 GradesHistogram (Array & Method)

Write a program called **GradesHistogram**, which reads in n grades (as in the previous exercise), and displays the horizontal and vertical histograms. For example:

```
 0 -  9: ***
10 - 19: ***
20 - 29:
30 - 39:
40 - 49: *
50 - 59: *****
60 - 69:
70 - 79:
80 - 89: *
90 -100: **


                  *
                  *
 *   *             *
 *   *             *             *
 *   *          *   *         *   *
```

```
0-9  10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100
```

10. Exercises on Command-line Arguments, Recursion

10.1 Arithmetic (Command-Line Arguments)

Write a program called **Arithmetic** that takes three command-line arguments: two integers followed by an arithmetic operator (+, -, * or /). The program shall perform the corresponding operation on the two integers and print the result. For example:

```
java Arithmetic 3 2 +
3+2=5

java Arithmetic 3 2 -
3-2=1

java Arithmetic 3 2 /
3/2=1
```

Hints

The method main(String[] args) takes an argument: "an array of String", which is often (but not necessary) named args. This parameter captures the command-line arguments supplied by the user when the program is invoked. For example, if a user invokes:

```
java Arithmetic 12345 4567 +
```

The three command-line arguments "12345", "4567" and "+" will be captured in a String array {"12345", "4567", "+"} and passed into the main() method as the argument args. That is,

```
args is: {"12345", "4567", "+"}  // args is a String array
args.length is: 3                // length of the array
args[0] is: "12345"              // 1st element of the String array
args[1] is: "4567"               // 2nd element of the String array
args[2] is: "+"                  // 3rd element of the String array
args[0].length() is: 5           // length of 1st String element
args[1].length() is: 4           // length of the 2nd String element
args[2].length() is: 1           // length of the 3rd String element
```

```
public class Arithmetic {
  public static void main (String[] args) {
    int operand1, operand2;
    char theOperator;

    // Check if there are 3 command-line arguments in the
    //  String[] args by using length variable of array.
    if (args.length != 3) {
      System.err.println("Usage: java Arithmetic int1 int2 op");
      return;
    }

    // Convert the 3 Strings args[0], args[1], args[2] to int and char.
    // Use the Integer.parseInt(aStr) to convert a String to an int.
    operand1 = Integer.parseInt(args[0]);
    operand2 = ......

    // Get the operator, assumed to be the first character of
    //  the 3rd string. Use method charAt() of String.
    theOperator = args[2].charAt(0);
    System.out.print(args[0] + args[2] + args[1] + "=");

    switch(theOperator) {
      case ('-'): System.out.println(operand1 - operand2); break;
      case ('+'): ......
      case ('*'): ......
      case ('/'): ......
      default:
        System.err.println("Error: invalid operator!");
    }
  }
}
```

Notes:

- To provide command-line arguments, use the "cmd" or "terminal" to run your program in the form "java ClassName arg1 arg2 ....".
- To provide command-line arguments in Eclipse, right click the source code ⇒ "Run As" ⇒ "Run Configurations..." ⇒ Select "Main" and choose the proper main class ⇒ Select "Arguments" ⇒ Enter the command-line arguments, e.g., "3 2 +" in "Program Arguments".
- To provide command-line arguments in NetBeans, right click the "Project" name ⇒ "Set Configuration" ⇒ "Customize..." ⇒ Select categories "Run" ⇒ Enter the command-line arguments, e.g., "3 2 +" in the "Arguments" box (but make sure you select the proper Main class).

Question: Try "java Arithmetic 2 4 *" (in CMD shell and Eclipse/NetBeans) and explain the result obtained. How to resolve this problem?

In Windows' CMD shell, * is known as a wildcard character, that expands to give the list of file in the directory (called Shell Expansion). For example, "dir *.java" lists all the file with extension of ".java". You could double-quote the * to prevent shell expansion. Eclipse has a bug in handling this, even * is double-quoted. NetBeans??

SumDigits (Command-line Arguments)

Write a program called **SumDigits** to sum up the individual digits of a positive integer, given in the command line. The output shall look like:

```
java SumDigits 12345
The sum of digits = 1 + 2 + 3 + 4 + 5 = 15
```

Exercises on Recursion

In programming, a recursive function (or method) calls itself. The classical example is factorial(n), which can be defined recursively as f(n)=n*f(n-1). Nonetheless, it is important to take note that a recursive function should have a terminating condition (or base case), in the case of factorial, f(0)=1. Hence, the full definition is:

```
factorial(n) = 1, for n = 0
factorial(n) = n * factorial(n-1), for all n > 1
```

For example, suppose n = 5:

```
// Recursive call
factorial(5) = 5 * factorial(4)
factorial(4) = 4 * factorial(3)
factorial(3) = 3 * factorial(2)
factorial(2) = 2 * factorial(1)
factorial(1) = 1 * factorial(0)
factorial(0) = 1  // Base case
// Unwinding
factorial(1) = 1 * 1 = 1
factorial(2) = 2 * 1 = 2
factorial(3) = 3 * 2 = 6
factorial(4) = 4 * 6 = 24
factorial(5) = 5 * 24 = 120 (DONE)
```

10.2 Factorial Recursive

Write a recursive method called factorial() to compute the factorial of the given integer.

```
public static int factorial(int n)
```

The recursive algorithm is:

```
factorial(n) = 1, if n = 0
factorial(n) = n * factorial(n-1), if n > 0
```

Compare your code with the iterative version of the factorial():

```
factorial(n) = 1*2*3*...*n
```

Hints

Writing recursive function is straight forward. You simply translate the recursive definition into code with return.

```
// Return the factorial of the given integer, recursively
public static int factorial(int n) {
   if (n == 0) {
      return 1;   // base case
   } else {
      return n * factorial(n-1);  // call itself
   }
   // or one liner
   // return (n == 0) ? 1 : n*factorial(n-1);
}
```

Notes

1. Recursive version is often much shorter.
2. The recursive version uses much more computational and storage resources, and it need to save its current states before each successive recursive call, so as to unwind later.

10.3 Fibonacci (Recursive)

Write a recursive method to compute the Fibonacci number of n, defined as follows:

```
F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)  for n >= 2
```

Compare the recursive version with the iterative version written earlier.

Hints

```
// Translate the recursive definition into code with return statements
public static int fibonacci(int n) {
   if (n == 0) {
      return 0;
   } else if (n == 1) {
      return 1;
   } else {
      return fibonacci(n-1) + fibonacci(n-2);
   }
}
```

10.4 Length of a Running Number Sequence (Recursive)

A special number sequence is defined as follows:

```
S(1) = 1
S(2) = 12
S(3) = 123
S(4) = 1234
......
S(9) = 123456789        // length is 9
```

```
S(10) = 12345678910      // length is 11
S(11) = 1234567891011    // length is 13
S(12) = 123456789101112  // length is 15
......
```

Write a recursive method to compute the length of S(n), defined as follows:

```
len(1) = 1
len(n) = len(n-1) + numOfDigits(n)
```

Also write an iterative version.

10.5 GCD (Recursive)

Write a recursive method called gcd() to compute the greatest common divisor of two given integers.

```
public static void int gcd(int a, int b)


gcd(a,b) = a, if b = 0
gcd(a,b) = gcd(b, remainder(a,b)), if b > 0
```

11. More (Difficult) Exercises

11.1 JDK Source Code

Extract the source code of the class Math from the JDK source code (JDK Installed Directory ⇒ "lib" ⇒ "src.zip" ⇒ "java.base" ⇒ "java" ⇒ "lang" ⇒ "Math.java"). Study how constants such as E and PI are defined. Also study how methods such as abs(), max(), min(), toDegree(), etc, are written.

Also study the "Integer.java", "String.java".

11.2 Matrices (2D Arrays)

Similar to Math class, write a Matrix library that supports matrix operations (such as addition, subtraction, multiplication) via 2D arrays. The operations shall support both double and int. Also write a test class to exercise all the operations programmed.

Hints

```
public class Matrix {
   // Method signatures
   public static void print(int[][] m);
   public static void print(double[][] m);
   public static boolean haveSameDimension(int[][] m1, int[][] m2);  // Used in add(), subtract()
   public static boolean haveSameDimension(double[][] m1, double[][] m2);
   public static int[][] add(int[][] m1, int[][] m2);
   public static double[][] add(double[][] m1, double[][] m2);
   public static int[][] subtract(int[][] m1, int[][] m2);
   public static double[][] subtract(double[][] m1, double[][] m2);
   public static int[][] multiply(int[][] m1, int[][] m2);
   public static double[][] multiply(double[][] m1, double[][] m2);
   ......
}
```

11.3 PrintAnimalPattern (Special Characters and Escape Sequences)

Write a program called **PrintAnimalPattern**, which uses println() to produce this pattern:

```
          '__'
          (©©)
  /========\/
 / || %% ||
*  ||----||
¥¥   ¥¥
""   ""
```

Hints

Use escape sequence \uhhhh where hhhh are four hex digits to display Unicode characters such as ¥ and ©. ¥ is 165 (00A5H) and © is 169 (00A9H) in both ISO-8859-1 (Latin-1) and Unicode character sets.

Double-quote (") and black-slash (\) require escape sequence inside a String. Single quote (') does not require escape sign.

Try

Print the same pattern using printf(). (Hints: Need to use %% to print a % in printf() because % is the suffix for format specifier.)

11.4 Print Patterns (nested-loop)

Write a method to print each of the followings patterns using nested loops in a class called **PrintPatterns**. The program shall prompt user for the sizde of the pattern. The signatures of the methods are:

```
public static void printPatternX(int size);  // X: A, B, C,...; size is a positive integer.
```

```
# # # # # # # # # # #           #                 #
 # # # # # # # # #           # # #              # # #
   # # # # # # #          # # # # #            # # # # #
    # # # # #           # # # # # # #         # # # # # # #
     # # #           # # # # # # # # #      # # # # # # # # #
      #           # # # # # # # # # # #    # # # # # # # # # # #
      (a)               (b)           # # # # # # # # #
                                      # # # # # # #
                                        # # # # #
                                         # # #
                                          #
                                         (c)

1             1 2 3 4 5 6 7 8               1   8 7 6 5 4 3 2 1
1 2             1 2 3 4 5 6 7             2 1    7 6 5 4 3 2 1
1 2 3             1 2 3 4 5 6           3 2 1    6 5 4 3 2 1
1 2 3 4             1 2 3 4 5         4 3 2 1     5 4 3 2 1
1 2 3 4 5             1 2 3 4       5 4 3 2 1      4 3 2 1
1 2 3 4 5 6             1 2 3     6 5 4 3 2 1      3 2 1
1 2 3 4 5 6 7             1 2   7 6 5 4 3 2 1       2 1
1 2 3 4 5 6 7 8             1 8 7 6 5 4 3 2 1        1
   (d)            (e)             (f)          (g)



         1              1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
        1 2 1             1 2 3 4 5 6 7 6 5 4 3 2 1
      1 2 3 2 1              1 2 3 4 5 6 5 4 3 2 1
     1 2 3 4 3 2 1             1 2 3 4 5 4 3 2 1
    1 2 3 4 5 4 3 2 1             1 2 3 4 3 2 1
   1 2 3 4 5 6 5 4 3 2 1            1 2 3 2 1
```

```
  1 2 3 4 5 6 7 6 5 4 3 2 1                    1 2 1
1 2 3 4 5 6 7 8 7 6 5 4 3 2 1                    1
             (h)                           (i)

1                          1      1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
1 2                      2 1      1 2 3 4 5 6 7   7 6 5 4 3 2 1
1 2 3                  3 2 1      1 2 3 4 5 6       6 5 4 3 2 1
1 2 3 4              4 3 2 1      1 2 3 4 5           5 4 3 2 1
1 2 3 4 5          5 4 3 2 1      1 2 3 4               4 3 2 1
1 2 3 4 5 6      6 5 4 3 2 1      1 2 3                   3 2 1
1 2 3 4 5 6 7  7 6 5 4 3 2 1      1 2                       2 1
1 2 3 4 5 6 7 8 7 6 5 4 3 2 1      1                          1
             (j)                           (k)

              1
            2 3 2
          3 4 5 4 3
        4 5 6 7 6 5 4
      5 6 7 8 9 8 7 6 5
    6 7 8 9 0 1 0 9 8 7 6
  7 8 9 0 1 2 3 2 1 0 9 8 7
8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
             (l)
```

11.5 Print Triangles (nested-loop)

Write a method to print each of the following patterns using nested-loops in a class called **PrintTriangles**. The program shall prompt user for the number of rows. The signatures of the methods are:

```
public static void printXxx(int numRows);  // Xxx is the pattern's name
```

```
                           1
                       1   2   1
                   1   2   4   2   1
               1   2   4   8   4   2   1
           1   2   4   8  16   8   4   2   1
       1   2   4   8  16  32  16   8   4   2   1
   1   2   4   8  16  32  64  32  16   8   4   2   1
1   2   4   8  16  32  64 128  64  32  16   8   4   2   1
                 (a) PowerOf2Triangle

1                                     1
1  1                                1   1
1  2  1                           1   2   1
1  3  3  1                      1   3   3   1
1  4  6  4  1                 1   4   6   4   1
1  5 10 10  5  1            1   5  10  10   5   1
1  6 15 20 15  6  1       1   6  15  20  15   6   1
(b) PascalTriangle1          (c) PascalTriangle2
```

11.6 Trigonometric Series

Write a method to compute sin(x) and cos(x) using the following series expansion, in a class called **TrigonometricSeries**. The signatures of the methods are:

```
public static double sin(double x, int numTerms);   // x in radians, NOT degrees
public static double cos(double x, int numTerms);
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Compare the values computed using the series with the JDK methods Math.sin(), Math.cos() at x=0, π/6, π/4, π/3, π/2 using various numbers of terms.

Hints

Do not use int to compute the factorial; as factorial of 13 is outside the int range. Avoid generating large numerator and denominator. Use double to compute the terms as:

$$\frac{x^n}{n!} = \left(\frac{x}{n}\right)\left(\frac{x}{n-1}\right)\cdots\left(\frac{x}{1}\right)$$

11.7 Exponential Series

Write a method to compute e and exp(x) using the following series expansion, in a class called **ExponentialSeries**. The signatures of the methods are:

```
public static double exp(int numTerms);   // x in radians
public static double exp(double x, int numTerms);
```

$$\mathrm{e} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

$$\mathrm{e}^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

11.8 Special Series

Write a method to compute the sum of the series in a class called SpecialSeries. The signature of the method is:

```
public static double specialSeries(double x, int numTerms);
```

$$x + \frac{1}{2} \times \frac{x^3}{3} + \frac{1 \times 3}{2 \times 4} \times \frac{x^5}{5} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{x^7}{7} + \frac{1 \times 3 \times 5 \times 7}{2 \times 4 \times 6 \times 8} \times \frac{x^9}{9} + \cdots; \quad -1 \le x \le 1$$

11.9 FactorialInt (Handling Overflow)

Write a program called **FactorialInt** to list all the factorials that can be expressed as an int (i.e., 32-bit signed integer in the range of [-2147483648, 2147483647]). Your output shall look like:

```
The factorial of 1 is 1
The factorial of 2 is 2
...
The factorial of 12 is 479001600
The factorial of 13 is out of range
```

Hints

The maximum and minimum values of a 32-bit int are kept in constants Integer.MAX_VALUE and Integer.MIN_VALUE, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use F(n) * (n+1) > Integer.MAX_VALUE to check for overflow. Instead, overflow occurs for F(n+1) if (Integer.MAX_VALUE / Factorial(n)) < (n+1), i.e., no more room for the next number.

Try

Modify your program called **FactorialLong** to list all the factorial that can be expressed as a long (64-bit signed integer). The maximum value for long is kept in a constant called Long.MAX_VALUE.

11.10 FibonacciInt (Handling Overflow)

Write a program called **FibonacciInt** to list all the Fibonacci numbers, which can be expressed as an int (i.e., 32-bit signed integer in the range of [-2147483648, 2147483647]). The output shall look like:

```
F(0) = 1
F(1) = 1
F(2) = 2
...
F(45) = 1836311903
F(46) is out of the range of int
```

Hints

The maximum and minimum values of a 32-bit int are kept in constants Integer.MAX_VALUE and Integer.MIN_VALUE, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use F(n) = F(n-1) + F(n-2) > Integer.MAX_VALUE to check for overflow. Instead, overflow occurs for F(n) if Integer.MAX_VALUE – F(n-1) < F(n-2) (i.e., no more room for the next Fibonacci number).

Try

Write a similar program called **TribonacciInt** for Tribonacci numbers.

11.11 Number System Conversion

Write a method call **toRadix()** which converts a positive integer from one radix into another. The method has the following header:

```
public static String toRadix(String in, int inRadix, int outRadix)  // The input and output are treated as String.
```

Write a program called **NumberConversion**, which prompts the user for an input string, an input radix, and an output radix, and display the converted number. The output shall look like:

```
Enter a number and radix: A1B2
Enter the input radix: 16
Enter the output radix: 2
"A1B2" in radix 16 is "1010000110110010" in radix 2.
```

11.12 NumberGuess

Write a program called **NumberGuess** to play the number guessing game. The program shall generate a random number between 0 and 99. The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly. For example:

```
java NumberGuess
Key in your guess:
50
Try higher
70
Try lower
65
Try lower
61
You got it in 4 trials!
```

Hints

Use Math.random() to produce a random number in double between 0.0 (inclusive) and 1.0 (exclusive). To produce an int between 0 and 99, use:

```
final int SECRET_NUMBER = (int)(Math.random()*100);  // truncate to int
```

11.13 WordGuess

Write a program called **WordGuess** to guess a word by trying to guess the individual characters. The word to be guessed shall be provided using the command-line argument. Your program shall look like:

```
java WordGuess testing
Key in one character or your guess word: t
Trial 1: t__t___
Key in one character or your guess word: g
Trial 2: t__t__g
Key in one character or your guess word: e
Trial 3: te_t__g
Key in one character or your guess word: testing
Congratulation!
You got in 4 trials
```

Hints

Set up a boolean array (of the length of the word to be guessed) to indicate the positions of the word that have been guessed correctly.

Check the length of the input String to determine whether the player enters a single character or a guessed word. If the player enters a single character, check it against the word to be guessed, and update the boolean array that keeping the result so far.

Try

Try retrieving the word to be guessed from a text file (or a dictionary) randomly.

11.14 DateUtil

Complete the following methods in a class called **DateUtil**:

- boolean isLeapYear(int year): returns true if the given year is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.
- boolean isValidDate(int year, int month, int day): returns true if the given year, month and day constitute a given date. Assume that year is between 1 and 9999, month is between 1 (Jan) to 12 (Dec) and day shall be between 1 and 28|29|30|31 depending on the month and whether it is a leap year.

- int getDayOfWeek(int year, int month, int day): returns the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT, for the given date. Assume that the date is valid.
- String toString(int year, int month, int day): prints the given date in the format "xxxday d mmm yyyy", e.g., "Tuesday 14 Feb 2012". Assume that the given date is valid.

Hints

To find the day of the week (Reference: Wiki "Determination of the day of the week"):

1700-	1800-	1900-	2000-	2100-	2200-	2300-	2400-
4	2	0	6	4	2	0	6

1. Based on the first two digit of the year, get the number from the following "century" table.
2. Take note that the entries 4, 2, 0, 6 repeat.
3. Add to the last two digit of the year.
4. Add to "the last two digit of the year divide by 4, truncate the fractional part".
5. Add to the number obtained from the following month table:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Non-Leap Year	0	3	3	6	1	4	6	2	5	0	3	5
Leap Year	6	2	same as above									

6. Add to the day.
7. The sum modulus 7 gives the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT.

For example: 2012, Feb, 17

```
(6 + 12 + 12/4 + 2 + 17) % 7 = 5 (Fri)
```

The skeleton of the program is as follows:

```
/* Utilities for Date Manipulation */
public class DateUtil {

  // Month's name – for printing
  public static String[] strMonths
    = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
      "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

  // Number of days in each month (for non-leap years)
  public static int[] daysInMonths
    = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

  // Returns true if the given year is a leap year
  public static boolean isLeapYear(int year) { ...... }

  // Return true if the given year, month, day is a valid date
  // year: 1-9999
  // month: 1(Jan)-12(Dec)
  // day: 1-28|29|30|31. The last day depends on year and month
  public static boolean isValidDate(int year, int month, int day) { ...... }

  // Return the day of the week, 0:Sun, 1:Mon, ..., 6:Sat
  public static int getDayOfWeek(int year, int month, int day) { ...... }

  // Return String "xxxday d mmm yyyy" (e.g., Wednesday 29 Feb 2012)
  public static String printDate(int year, int month, int day) { ...... }

  // Test Driver
  public static void main(String[] args) {
    System.out.println(isLeapYear(1900));  // false
    System.out.println(isLeapYear(2000));  // true
    System.out.println(isLeapYear(2011));  // false
    System.out.println(isLeapYear(2012));  // true

    System.out.println(isValidDate(2012, 2, 29));  // true
    System.out.println(isValidDate(2011, 2, 29));  // false
    System.out.println(isValidDate(2099, 12, 31)); // true
    System.out.println(isValidDate(2099, 12, 32)); // false

    System.out.println(getDayOfWeek(1982, 4, 24));  // 6:Sat
    System.out.println(getDayOfWeek(2000, 1, 1));   // 6:Sat
    System.out.println(getDayOfWeek(2054, 6, 19));  // 5:Fri
    System.out.println(getDayOfWeek(2012, 2, 17));  // 5:Fri

    System.out.println(toString(2012, 2, 14)); // Tuesday 14 Feb 2012
  }
}
```

Notes

You can compare the day obtained with the Java's Calendar class as follows:

```
// Construct a Calendar instance with the given year, month and day
Calendar cal = new GregorianCalendar(year, month - 1, day);  // month is 0-based
// Get the day of the week number: 1 (Sunday) to 7 (Saturday)
int dayNumber = cal.get(Calendar.DAY_OF_WEEK);
String[] calendarDays = { "Sunday", "Monday", "Tuesday", "Wednesday",
                          "Thursday", "Friday", "Saturday" };
// Print result
System.out.println("It is " + calendarDays[dayNumber - 1]);
```

The calendar we used today is known as Gregorian calendar, which came into effect in October 15, 1582 in some countries and later in other countries. It replaces the Julian calendar. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400. Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.

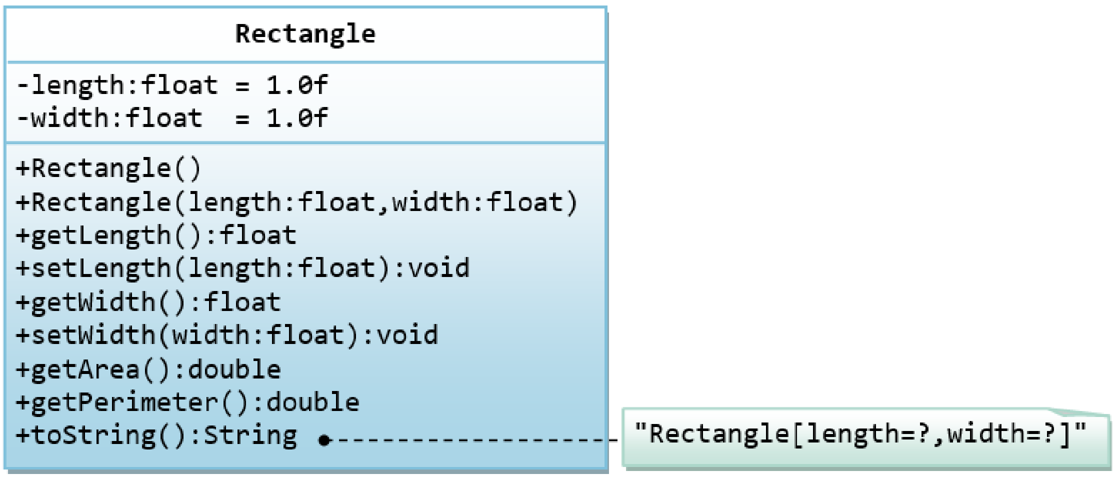
This above algorithm work for Gregorian dates only. It is difficult to modify the above algorithm to handle pre-Gregorian dates. A better algorithm is to find the number of days from a known date.

12. Exercises on Classes and Objects

12.1 The Rectangle Class

A class called Rectangle, which models a rectangle with a length and a width (in float), is designed as shown in the following class diagram. Write the Rectangle class.

Hints:



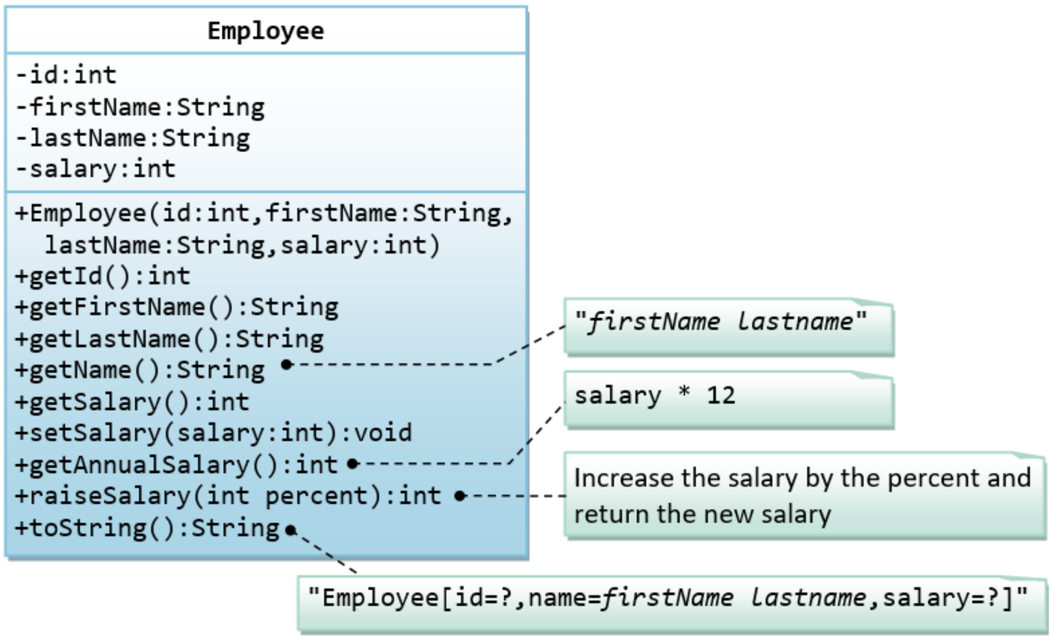
The expected output is:

```
Rectangle[length=1.2,width=3.4]
Rectangle[length=1.0,width=1.0]
Rectangle[length=5.6,width=7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

12.2 The Employee Class

A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method raiseSalary(percent) increases the salary by the given percentage. Write the Employee class.

Hints:


Employee
-id:int
-firstName:String
-lastName:String
-salary:int
+Employee(id:int,firstName:String, lastName:String,salary:int)
+getId():int
+getFirstName():String
+getLastName():String
+getName():String
+getSalary():int
+setSalary(salary:int):void
+getAnnualSalary():int
+raiseSalary(int percent):int
+toString():String
"firstName lastname"
salary * 12
Increase the salary by the percent and return the new salary
"Employee[id=?,name=firstName lastname,salary=?]"

The expected out is:

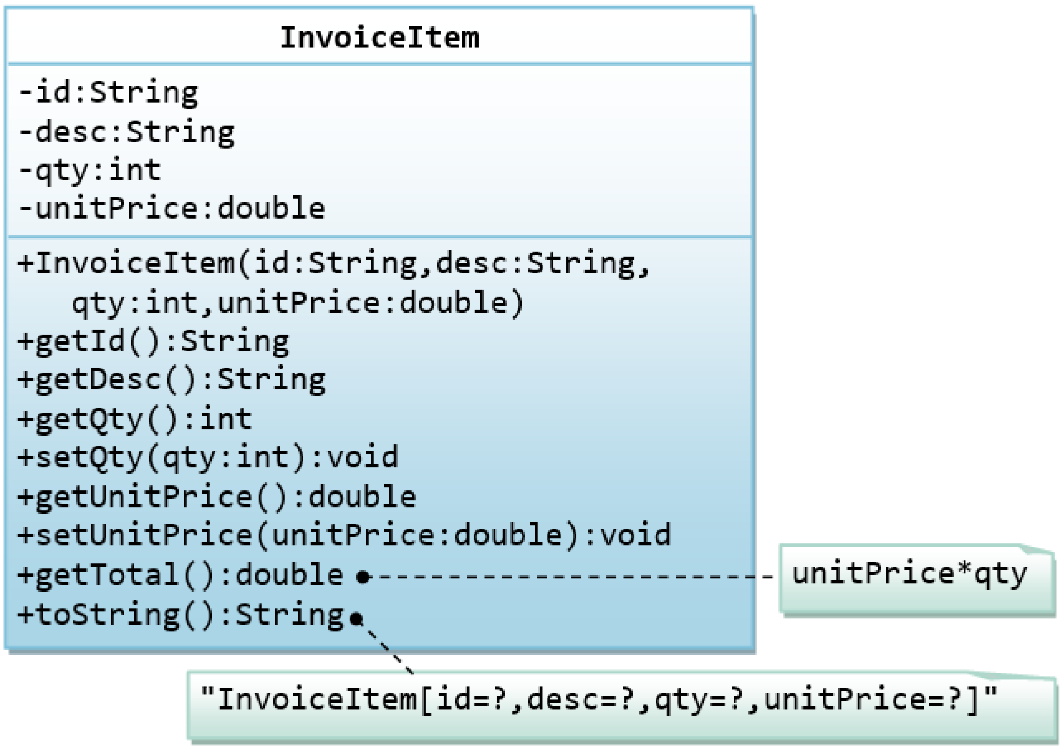
```
Employee[id=8,name=Peter Tan,salary=2500]
Employee[id=8,name=Peter Tan,salary=999]
id is: 8
```

```
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id=8,name=Peter Tan,salary=1098]
```

12.3 The InvoiceItem Class

A class called InvoiceItem, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. Write the InvoiceItem class.

Hints:



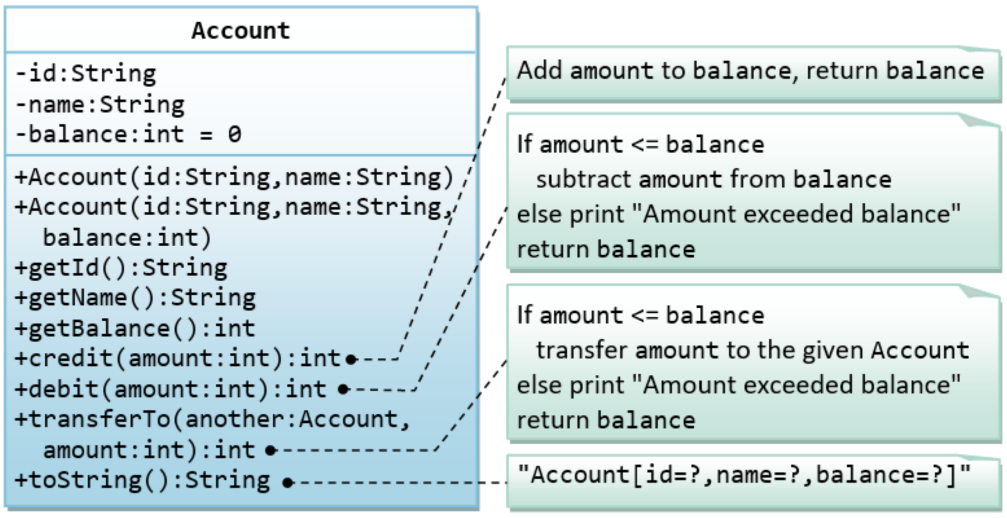
The expected output is:

```
InvoiceItem[id=A101,desc=Pen Red,qty=888,unitPrice=0.08]
InvoiceItem[id=A101,desc=Pen Red,qty=999,unitPrice=0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

12.4 The Account Class

A class called Account, which models a bank account of a customer, is designed as shown in the following class diagram. The methods credit(amount) and debit(amount) add or subtract the given amount to the balance. The method transferTo(anotherAccount, amount) transfers the given amount from this Account to the given anotherAccount. Write the Account class.

Hints:


Account
-id:String
-name:String
-balance:int = 0
+Account(id:String,name:String)
+Account(id:String,name:String, balance:int)
+getId():String
+getName():String
+getBalance():int
+credit(amount:int):int
+debit(amount:int):int
+transferTo(another:Account, amount:int):int
+toString():String
Add amount to balance, return balance
If amount <= balance subtract amount from balance else print "Amount exceeded balance" return balance
If amount <= balance transfer amount to the given Account else print "Amount exceeded balance" return balance
"Account[id=?,name=?,balance=?]"

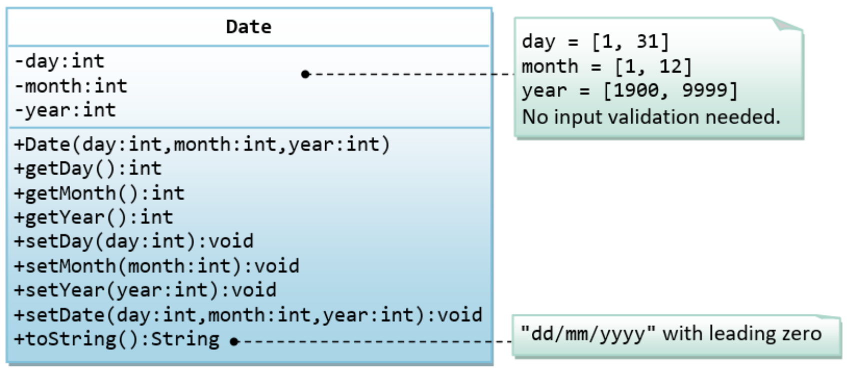
The expected output is:

```
Account[id=A101,name=Tan Ah Teck,balance=88]
Account[id=A102,name=Kumar,balance=0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id=A101,name=Tan Ah Teck,balance=188]
Account[id=A101,name=Tan Ah Teck,balance=138]
Amount exceeded balance
Account[id=A101,name=Tan Ah Teck,balance=138]
Account[id=A101,name=Tan Ah Teck,balance=38]
Account[id=A102,name=Kumar,balance=100]
```

12.5 The Date Class

A class called Date, which models a calendar date, is designed as shown in the following class diagram. Write the Date class.

Hints:



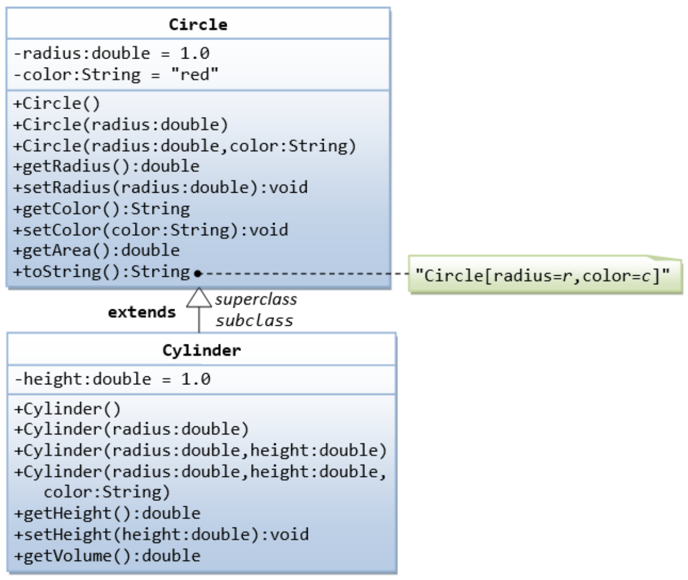
The expected output is:

```
01/02/2014
09/12/2099
Month: 12
Day: 9
Year: 2099
03/04/2016
```

13. Exercises on Inheritance

13.1 An Introduction to OOP Inheritance - The Circle and Cylinder Classes

This exercise shall guide you through the important concepts in inheritance.



In this exercise, a subclass called Cylinder is derived from the superclass Circle as shown in the class diagram (where an an arrow pointing up from the subclass to its superclass). Study how the subclass Cylinder invokes the superclass' constructors (via super() and super(radius)) and inherits the variables and methods from the superclass Circle.

You can reuse the Circle class that you have created in the previous exercise. Make sure that you keep "Circle.class" in the same directory.

```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // Constructor with default color, radius and height
   public Cylinder() {
      super();        // call superclass no-arg constructor Circle()
      height = 1.0;
   }
   // Constructor with default radius, color but given height
   public Cylinder(double height) {
      super();        // call superclass no-arg constructor Circle()
      this.height = height;
   }
   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);  // call superclass constructor Circle(r)
      this.height = height;
   }

   // A public method for retrieving the height
   public double getHeight() {
      return height;
   }

   // A public method for computing the volume of cylinder
   //  use superclass method getArea() to get the base area
   public double getVolume() {
      return getArea()*height;
   }
}
```

Method Overriding and "Super": The subclass Cylinder inherits getArea() method from its superclass Circle. Try overriding the getArea() method in the subclass Cylinder to compute the surface area (=2π×radius×height + 2×base-area) of the cylinder instead of base area. That is, if getArea() is called by a Circle instance, it returns the area. If getArea() is called by a Cylinder instance, it returns the surface area of the cylinder.

If you override the getArea() in the subclass Cylinder, the getVolume() no longer works. This is because the getVolume() uses the overridden getArea() method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the getVolume().

Hints: After overridding the getArea() in subclass Cylinder, you can choose to invoke the getArea() of the superclass Circle by calling super.getArea().

Try:

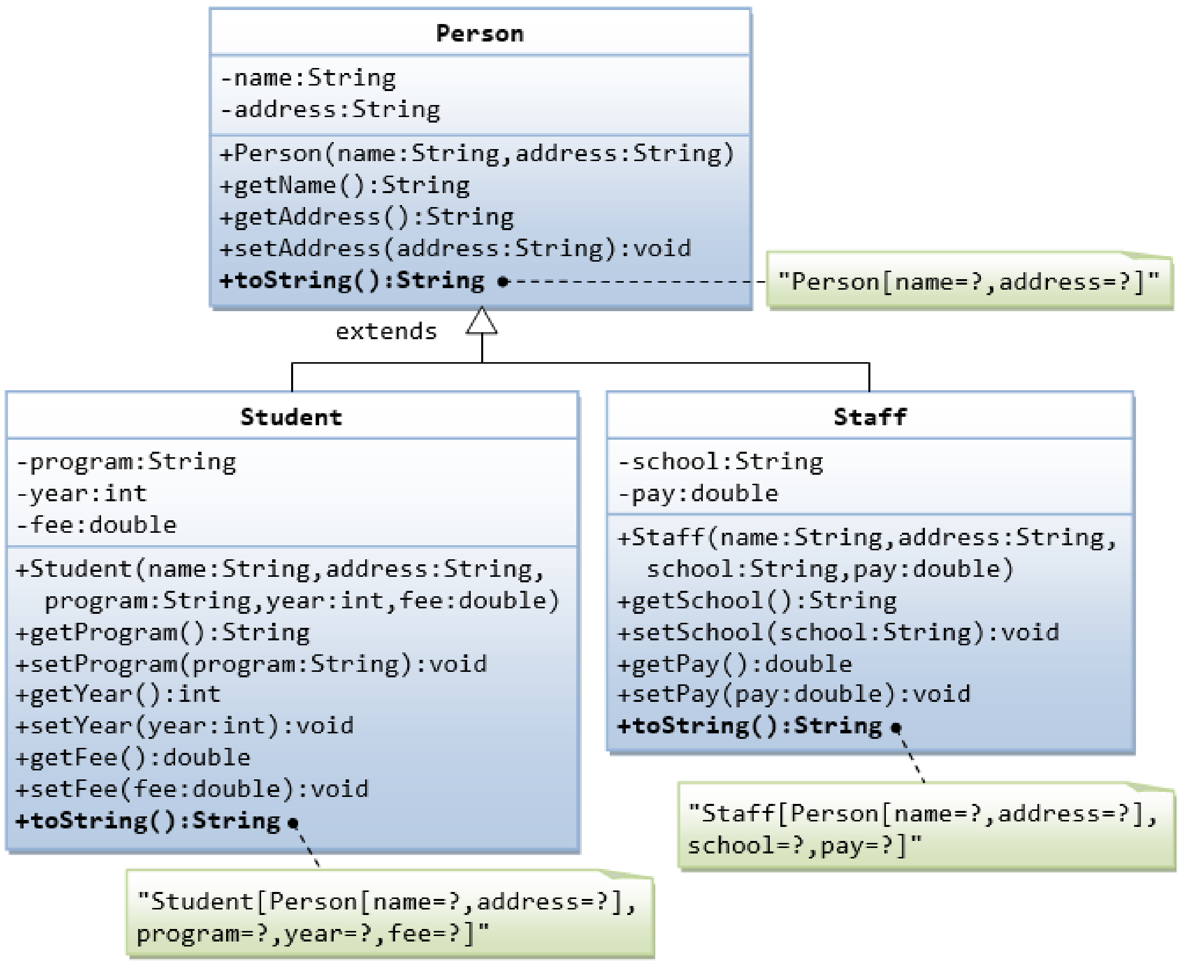
Provide a toString() method to the Cylinder class, which overrides the toString() inherited from the superclass Circle, e.g.,

```
@Override
public String toString() {      // in Cylinder class
   return "Cylinder: subclass of " + super.toString()  // use Circle's toString()
          + " height=" + height;
}
```

Try out the toString() method in TestCylinder.

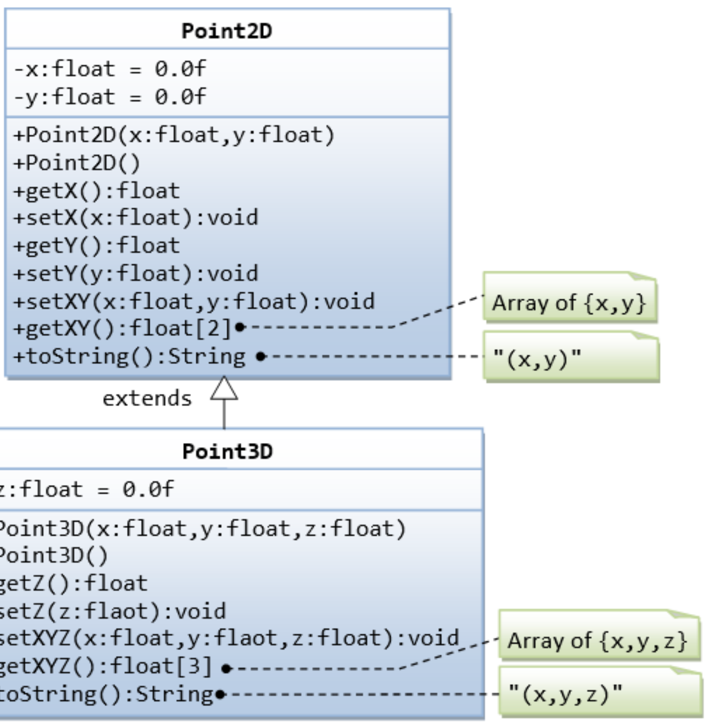
13.2 Superclass Person and its subclasses

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation @Override.


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
"Person[name=?,address=?]"
extends
Student
-program:String
-year:int
-fee:double
+Student(name:String,address:String, program:String,year:int,fee:double)
+getProgram():String
+setProgram(program:String):void
+getYear():int
+setYear(year:int):void
+getFee():double
+setFee(fee:double):void
+toString():String
"Student[Person[name=?,address=?], program=?,year=?,fee=?]"
Staff
-school:String
-pay:double
+Staff(name:String,address:String, school:String,pay:double)
+getSchool():String
+setSchool(school:String):void
+getPay():double
+setPay(pay:double):void
+toString():String
"Staff[Person[name=?,address=?], school=?,pay=?]"

13.3 Point2D and Point3D

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation @Override.



Hints:

1. You cannot assign floating-point literal say 1.1 (which is a double) to a float variable, you need to add a suffix f, e.g. 0.0f, 1.1f.
2. The instance variables x and y are private in Point2D and cannot be accessed directly in the subclass Point3D. You need to access via the public getters and setters. For example,

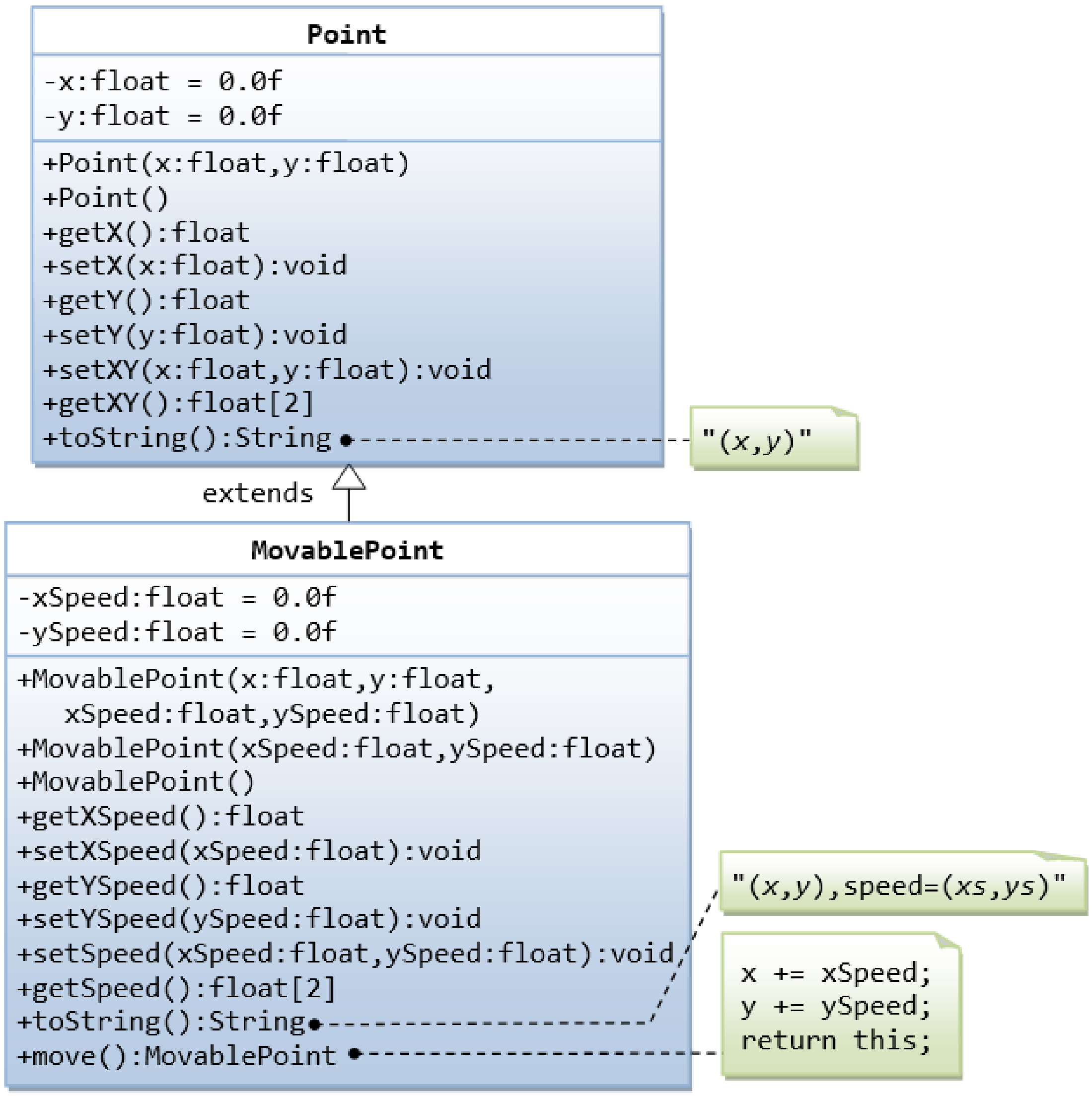
```
public void setXYZ(float x, float y, float z) {
      setX(x);     // or super.setX(x), use setter in superclass
      setY(y);
      this.z = z;
   }
```

3. The method getXY() shall return a float array:

```
public float[] getXY() {
      float[] result = new float[2];  // construct an array of 2 elements
      result[0] = ...
      result[1] = ...
      return result;  // return the array
   }
```

13.4 Point and MovablePoint

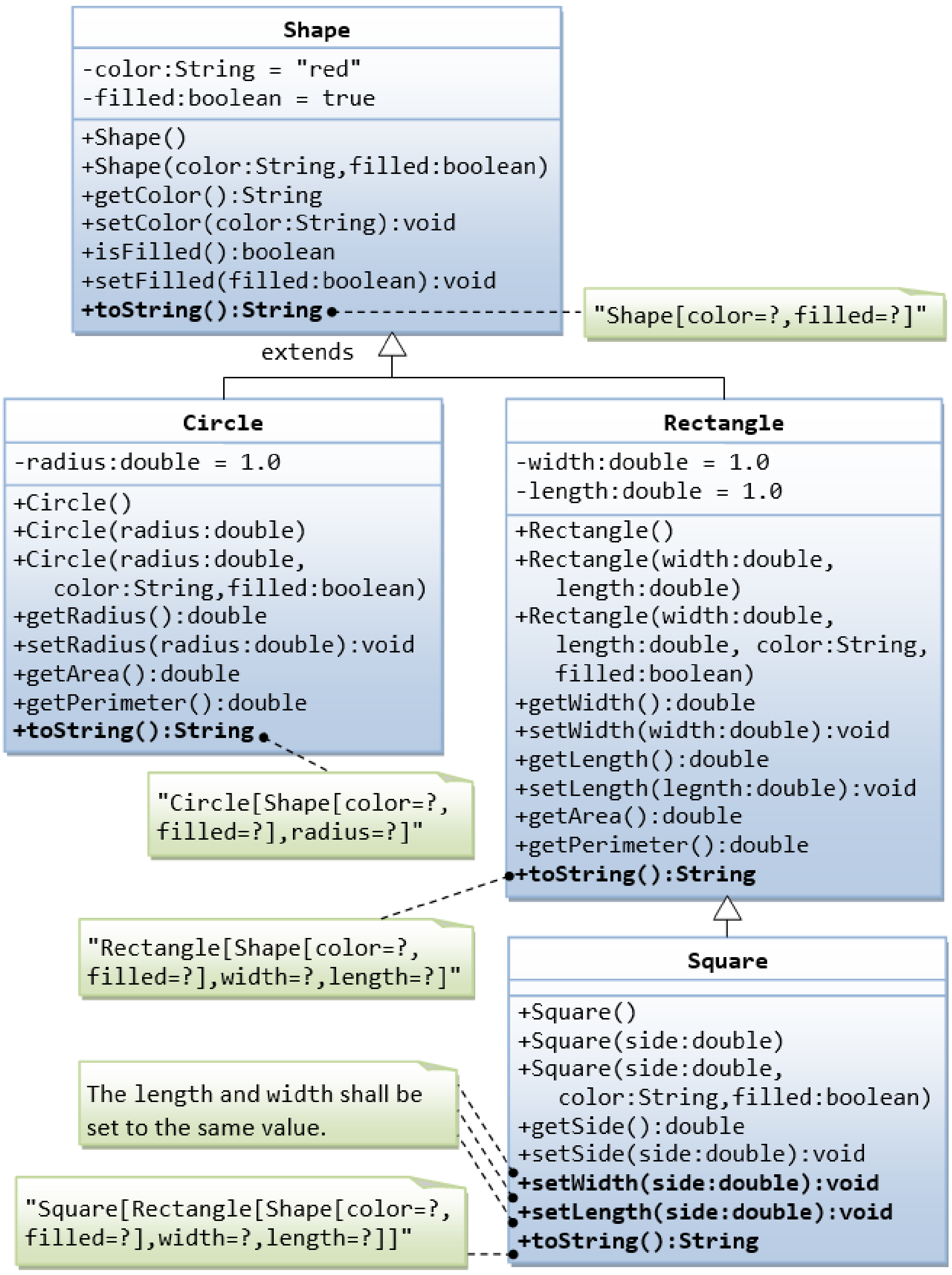
Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation @Override.



Hints

1. You cannot assign floating-point literal say 1.1 (which is a double) to a float variable, you need to add a suffix f, e.g. 0.0f, 1.1f.
2. The instance variables x and y are private in Point and cannot be accessed directly in the subclass MovablePoint. You need to access via the public getters and setters. For example, you cannot write x += xSpeed, you need to write setX(getX() + xSpeed).

13.5 Superclass Shape and its subclasses Circle, Rectangle and Square


Shape
-color:String = "red"
-filled:boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+toString():String
"Shape[color=?,filled=?]"
extends
Circle
-radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?, filled=?],radius=?]"
Rectangle
-width:double = 1.0
-length:double = 1.0
+Rectangle()
+Rectangle(width:double, length:double)
+Rectangle(width:double, length:double, color:String, filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?, filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double, color:String,filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
The length and width shall be set to the same value.
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

Write a superclass called Shape (as shown in the class diagram)
Write a test program to test all the methods defined in Shape.
Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.
Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram).

Hints:

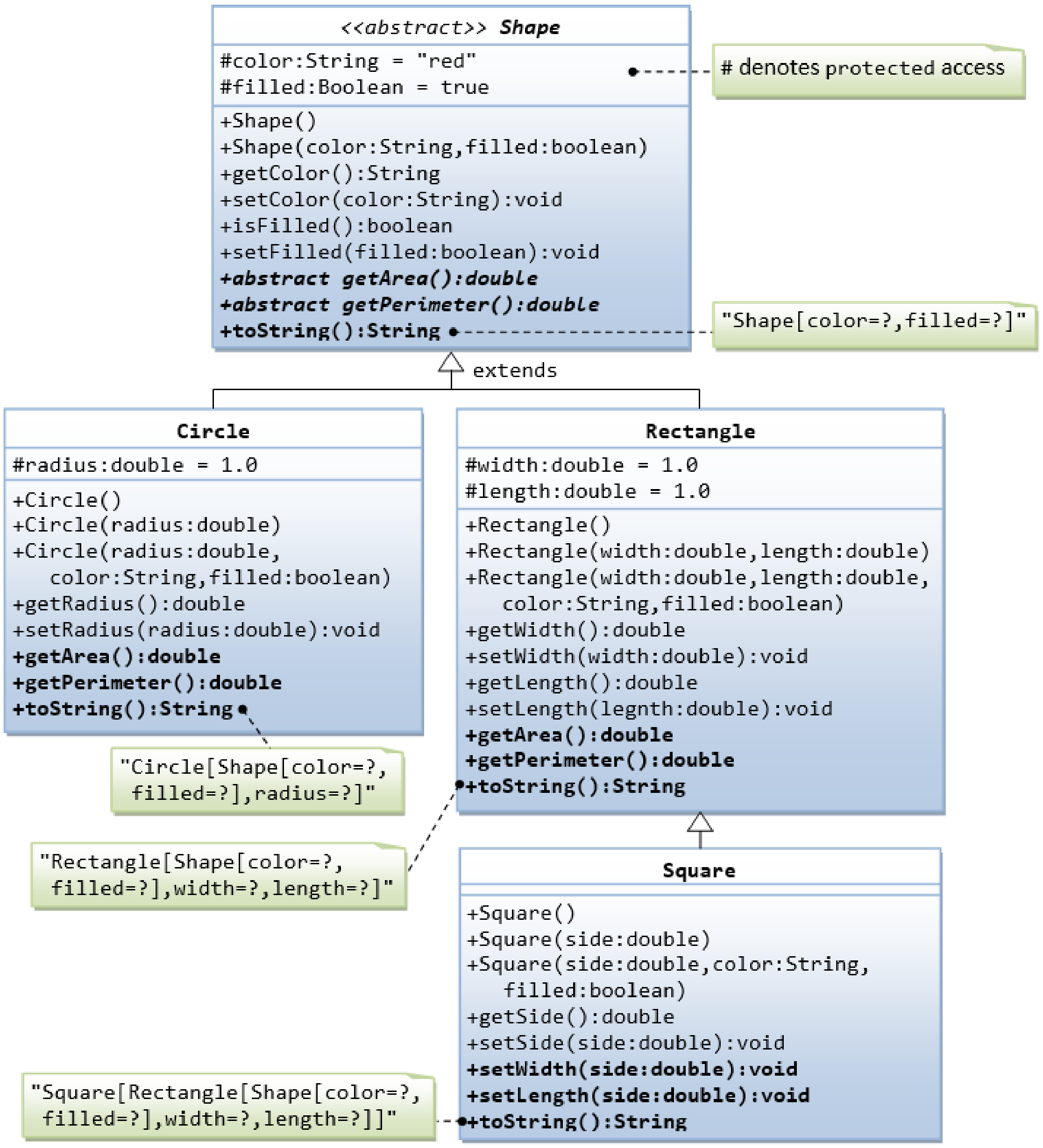
```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.
- Do you need to override the getArea() and getPerimeter()? Try them out.
- Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.

14. Exercises on Polymorphism, Abstract Classes and Interfaces

14.1 Ex: Abstract Superclass Shape and Its Concrete Subclasses

Rewrite the superclass Shape and its subclasses Circle, Rectangle and Square, as shown in the class diagram. Shape is an abstract class containing 2 abstract methods: getArea() and getPerimeter(), where its concrete subclasses must provide its implementation. All instance variables shall have protected access, i.e., accessible by its subclasses and classes in the same package. Mark all the overridden methods with annotation @Override.



<<abstract>> Shape
#color:String = "red"
#filled:Boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+abstract getArea():double
+abstract getPerimeter():double
+toString():String
# denotes protected access
"Shape[color=?,filled=?]"
extends
Circle
#radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?, filled=?],radius=?]"
Rectangle
#width:double = 1.0
#length:double = 1.0
+Rectangle()
+Rectangle(width:double,length:double)
+Rectangle(width:double,length:double, color:String,filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?, filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double,color:String, filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

In this exercise, Shape shall be defined as an abstract class, which contains:

- Two protected instance variables **color(String)** and **filled(boolean).** The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a '#' sign in the class diagram.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() (shown in italics in the class diagram).
- Subclasses Circle and Rectangle shall override the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also override the toString().

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

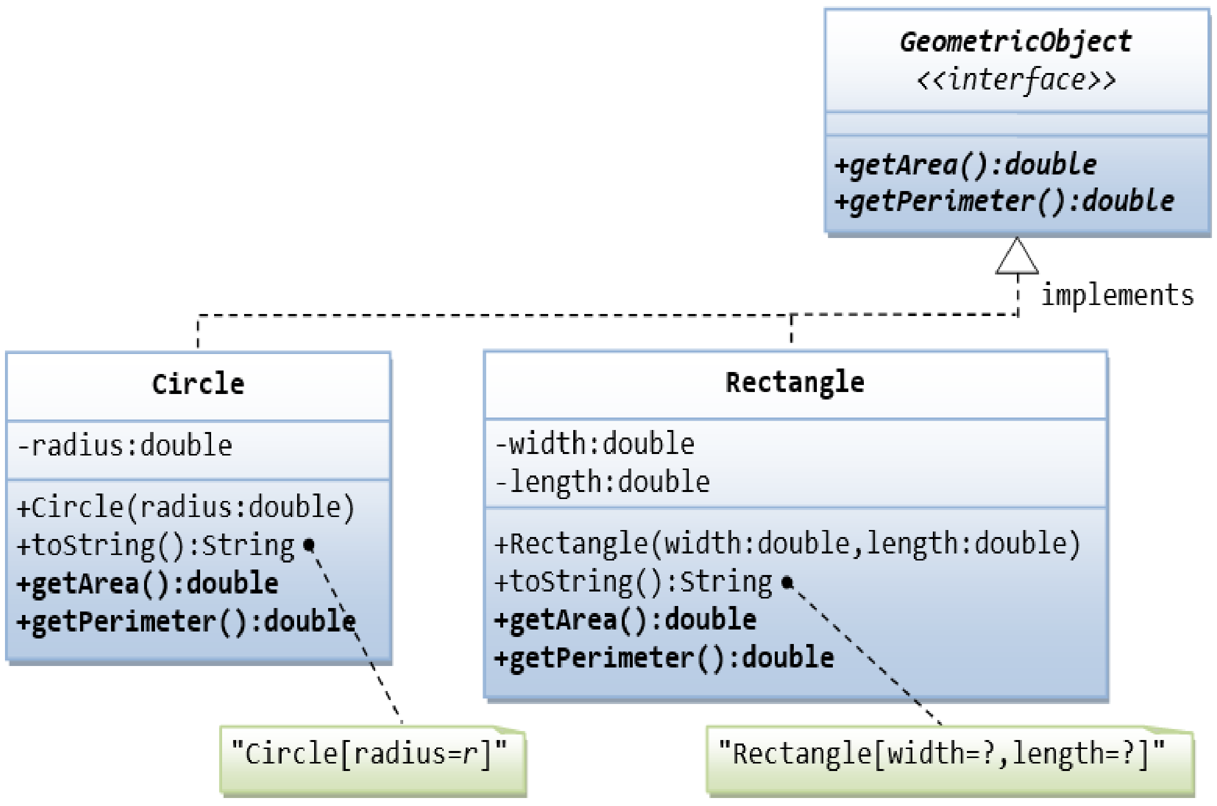
```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                    // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Try:

Explain the usage of the abstract method and abstract class?

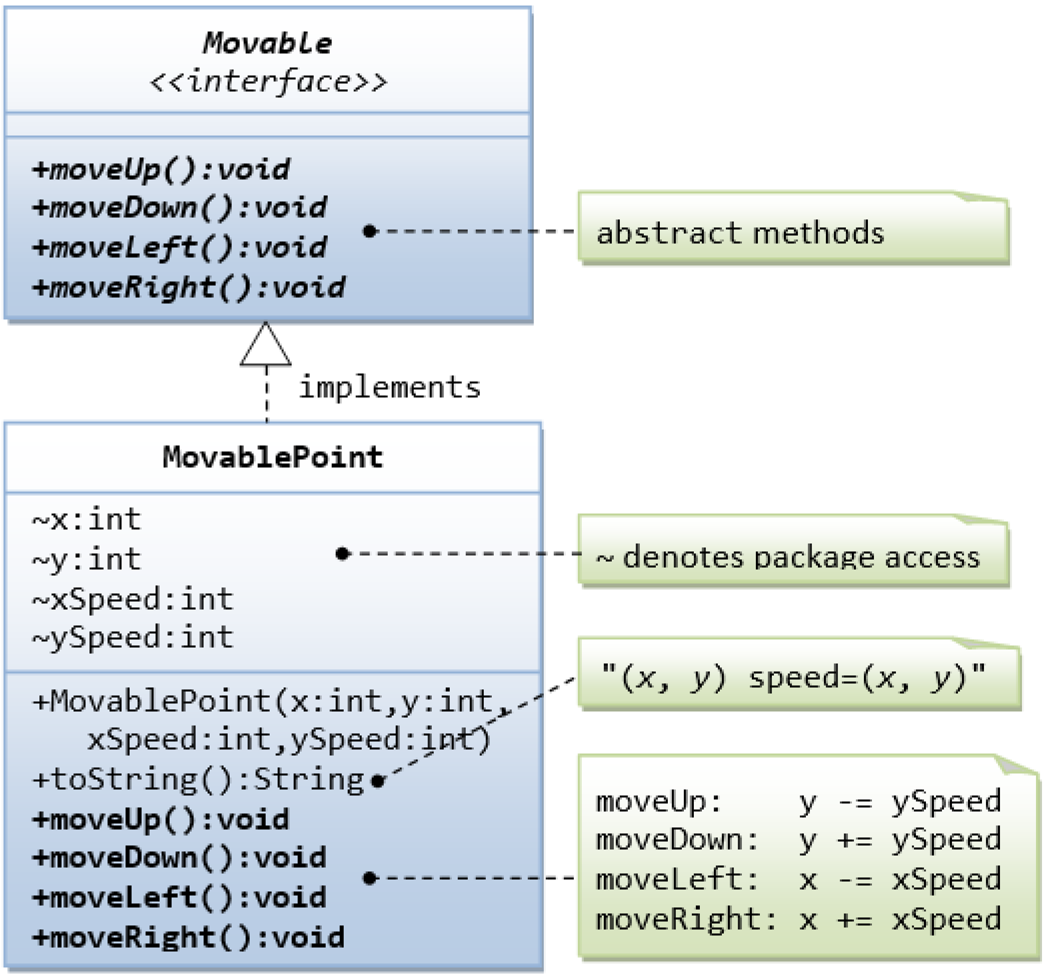
14.2 GeometricObject Interface and its Implementation Classes Circle and Rectangle

Write an interface called GeometricObject, which contains 2 abstract methods: getArea() and getPerimeter(), as shown in the class diagram. Also write an implementation class called Circle. Mark all the overridden methods with annotation @Override.



14.3 Ex: Movable Interface and its Implementation MovablePoint Class

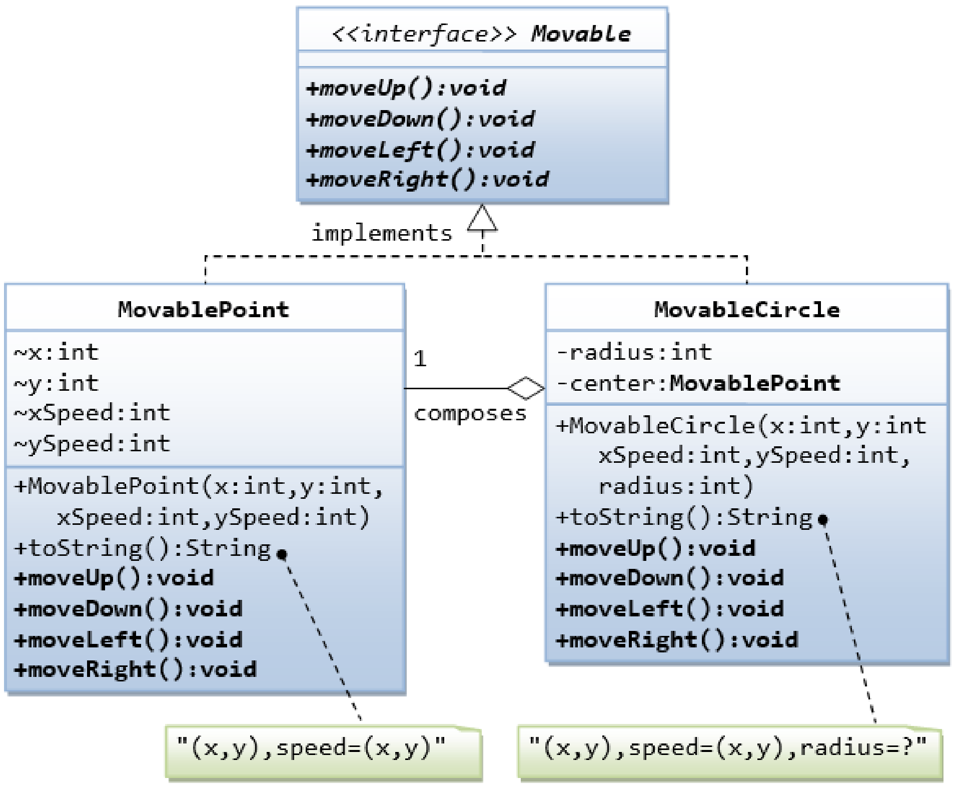
Write an interface called Movaable, which contains 4 abstract methods moveUp(), moveDown(), moveLeft() and moveRight(), as shown in the class diagram. Also write an implementation class called MovablePoint. Mark all the overridden methods with annotation @Override.



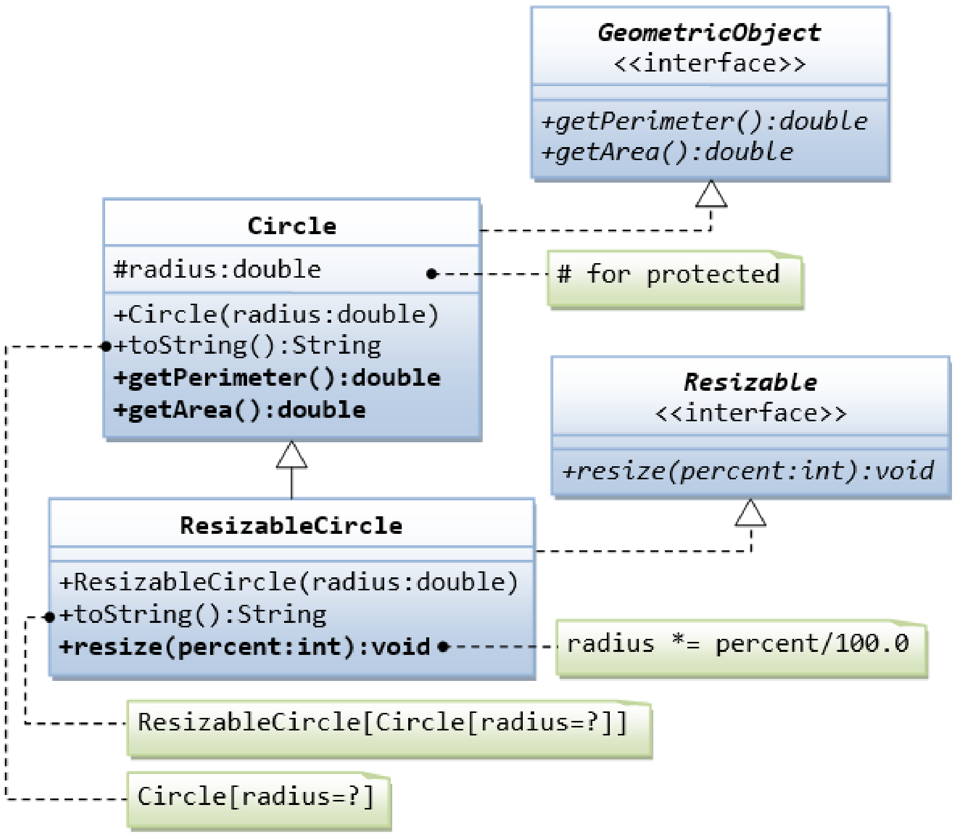
14.4 Movable Interface and Classes MovablePoint and MovableCircle

Write an interface called Movaable, which contains 4 abstract methods moveUp(), moveDown(),
moveLeft() and moveRight(),
as shown in the class diagram.

Write the implementation classes called MovablePoint and MovableCircle. Mark all the overridden methods with annotation @Override.



14.5 Interfaces Resizable and GeometricObject



Write the interface called GeometricObject, which declares two abstract methods: getParameter() and getArea(), as specified in the class diagram.

Hints:

```
public interface GeometricObject {
  public double getPerimeter();
  ......
}
```

Write the implementation class Circle, with a protected variable radius, which implements the interface GeometricObject.

Hints:

```
public class Circle implements GeometricObject {
  // Private variable
  ......

  // Constructor
  ......

  // Implement methods defined in the interface GeometricObject
  @Override
  public double getPerimeter() { ...... }

  ......
}
```

Write a test program called TestCircle to test the methods defined in Circle.

The class ResizableCircle is defined as a subclass of the class Circle, which also implements an interface called Resizable, as shown in class diagram. The interface Resizable declares an abstract method resize(), which modifies the dimension (such as radius) by the given percentage. Write the interface Resizable and the class ResizableCircle.

Hints:

```
public interface Resizable {
  public double resize(...);
}
```

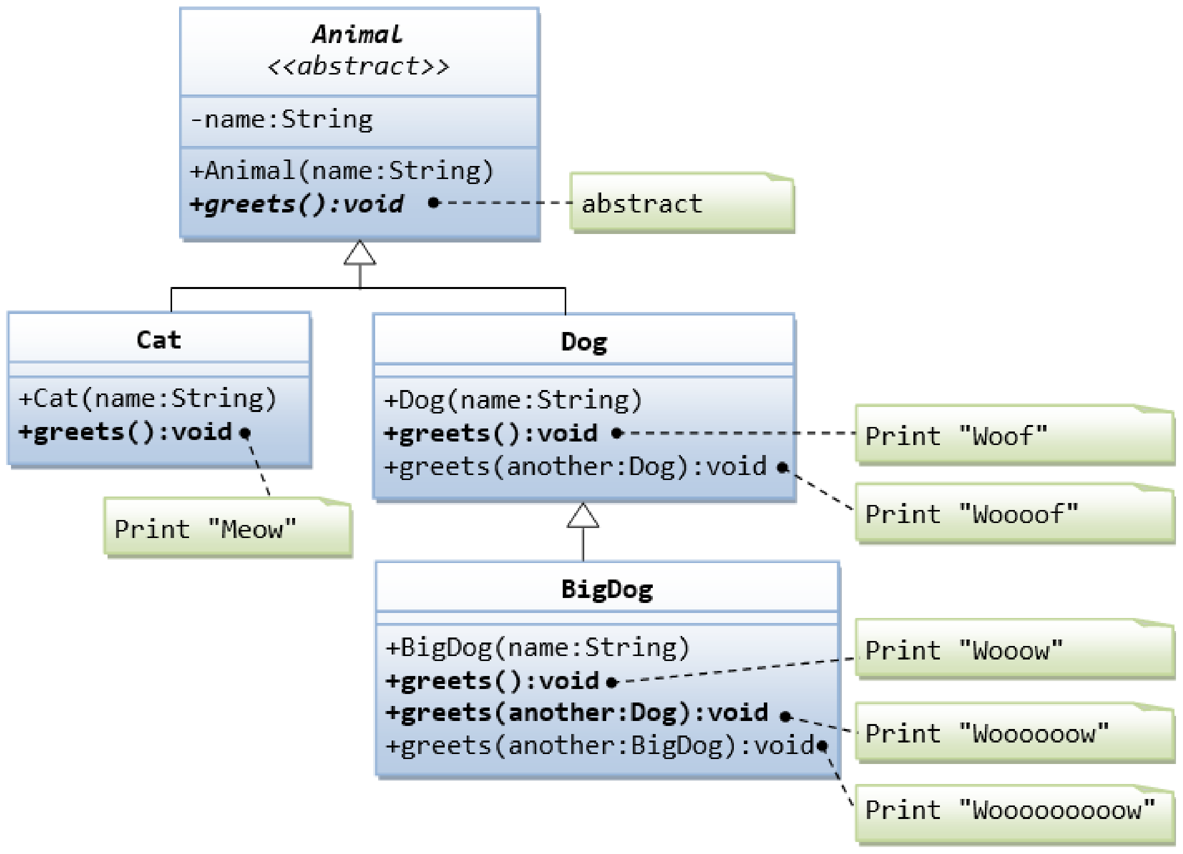
```
public class ResizableCircle extends Circle implements Resizeable {

  // Constructor
  public ResizableCircle(double radius) {
    super(...);
  }

  // Implement methods defined in the interface Resizable
  @Override
  public double resize(int percent) { ...... }
}
```

Try:

Write a test program called TestResizableCircle to test the methods defined in ResizableCircle.

14.6 Abstract Superclass Animal and its Implementation Subclasses

Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.



14.7 Another View of Abstract Superclass Animal and its Implementation Subclasses

Examine the following codes and draw the class diagram.

```
abstract public class Animal {
   abstract public void greeting();
}
```

```
public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}
public class Dog extends Animal {
   @Override
   public void greeting() {
      System.out.println("Woof!");
   }

   public void greeting(Dog another) {
      System.out.println("Woooooooooof!");
   }
}
```

```
public class BigDog extends Dog {
   @Override
   public void greeting() {
      System.out.println("Woow!");
   }

   @Override
   public void greeting(Dog another) {
      System.out.println("Woooooowwwww!");
   }
}
```

Try:
Explain the outputs (or error) for the following test program.

```
public class TestAnimal {
   public static void main(String[] args) {
      // Using the subclasses
      Cat cat1 = new Cat();
      cat1.greeting();
      Dog dog1 = new Dog();
      dog1.greeting();
      BigDog bigDog1 = new BigDog();
      bigDog1.greeting();

      // Using Polymorphism
      Animal animal1 = new Cat();
      animal1.greeting();
      Animal animal2 = new Dog();
      animal2.greeting();
      Animal animal3 = new BigDog();
      animal3.greeting();
      Animal animal4 = new Animal();

      // Downcast
      Dog dog2 = (Dog)animal2;
      BigDog bigDog2 = (BigDog)animal3;
      Dog dog3 = (Dog)animal3;
      Cat cat2 = (Cat)animal2;
      dog2.greeting(dog3);
      dog3.greeting(dog2);
      dog2.greeting(bigDog2);
      bigDog2.greeting(dog2);
      bigDog2.greeting(bigDog1);
   }
}
```

15. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

	Name of the contest	Minimum number of problems to solve	Required score
•	CodeChef	20	200
•	Leetcode	20	200
•	GeeksforGeeks	20	200
•	SPOJ	5	50
•	InterviewBit	10	1000
•	Hackerrank	25	250
•	Codeforces	10	100
•	BuildIT	50	500
		Total score need to obtain	2500

Student must have any one of the following certifications:

- HackerRank – Java Basic Skills Certification
- Oracle Certified Associate Java Programmer OCAJP
- CodeChef - Learn Java Certification
- NPTEL – Programming in Java
- NPTEL – Data Structures and Algorithms in Java

V. TEXT BOOKS:

1. Farrell, Joyce.Java Programming, Cengage Learning B S Publishers, 8th Edition, 2020
2. Schildt, Herbert. Java: The Complete Reference 11th Edition, McGraw-Hill Education, 2018.

VI. REFERENCE BOOKS:

1. Deitel, Paul and Deitel, Harvey. Java: How to Program, Pearson, 11th Edition, 2018.
2. Evans, Benjamin J. and Flanagan, David. Java in a Nutshell, O'Reilly Media, 7th Edition, 2018.
3. Bloch, Joshua. Effective Java, Addison-Wesley Professional, 3rd Edition, 2017.
4. Sierra, Kathy and Bates, Bert. Head First Java, O'Reilly Media, 2nd Edition, 2005

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/en/java/
2. https://www.geeksforgeeks.org/java
3. https://www.tutorialspoint.com/java/index.htm
4. https://www.coursera.org/courses?query=java

VIII. MATERIALS ONLINE;

1. Syllabus
2. Lab manual

COURSE CONTENT

MANUFACTURING PRACTICE								
I Semester: CSE (AI & ML) / IT / ECE / EEE **II Semester:** CSE / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AMED02	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	1	2	2	40	60	100
Contact Classes: Nil	**Tutorial Classes: 15**	**Practical Classes:30**				**Total Classes:45**		
Prerequisite: There is no prerequisite for this course.								

I. COURSE OVERVIEW:

This course provides the opportunity to become confident with new tools, equipment, and techniques for creating physical objects and mechanisms with a variety of materials. The students will learn the concepts of 3D printing, laser cutting, circuit board soldering, wood carving and CNC machining. Skills learned in the course enable the students about the design process in digital manufacturing used in various industrial applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The digital and additive manufacturing techniques used in various industrial applications in the current era to develop prototype models.
II. The unconventional machining processes and their selective applications as an alternative to traditional manufacturing methods.
III. The standard electrical wiring practices for domestic and industrial appliances.
IV. The soldering and de-soldering components on a circuit board safely and correctly.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Practice the various types of manufacturing methods for preparing the given material to desired shape by using traditional and unconventional manufacturing practices.
CO 2	Execute the additive manufacturing technology for learning about the 3D printing processes and techniques.
CO 3	Select computer numerical control laser techniques for preparing the required geometrical profiles on non-metallic materials.
CO 4	Demonstrate the assembly and disassembly of electrical equipment's and controls for safe domestic applications.
CO 5	Make use of computer numerical technologies to create products using wood carving techniques.
CO 6	Apply the plumbing skills to work with fittings and pipes made of PVC and galvanized steel.

IV. COURSE CONTENT:

EXERCISES ON MANUFACTURING PRACTICES

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

All dimensions are in mm in experiments.

Safety

Safety is a vital issue in all workplaces. Before using any equipment and machines or attempt practical work in a workshop everyone must understand basic safety rules. These rules will help keep all safe in the workshop.

Safety Rules

1. Always listen carefully to the teacher and follow instructions.
2. When learning how to use a machine, listen very carefully to all the instructions given by the faculty / instructor. Ask questions, especially if you do not fully understand.
3. Always wear an apron as it will protect your clothes and holds lose clothing such as ties in place.
4. Wear good strong shoes.
5. Bags should not be brought into a workshop as people can trip over them.
6. Do not use a machine if you have not been shown how to operate it safely by the faculty / instructors
7. Know where the emergency stop buttons are positioned in the workshop. If you see an accident at the other side of the workshop you can use the emergency stop button to turn off all electrical power to machines.
8. Wherever required, wear protective equipment, such as goggles, safety glasses, masks, gloves, hair nets, etc.
9. Always be patient, never rush in the workshop.
10. Always use a guard when working on a machine.
11. Keep hands away from moving/rotating machinery.
12. Use hand tools carefully, keeping both hands behind the cutting edge.
13. Report any UNSAFE condition or acts to instructor.
14. Report any damage to machines/equipment as this could cause an accident.
15. Keep your work area clean.

1. Getting Started Exercises

1.1 Principles of 3D printing and additive manufacturing techniques

3D printing or additive manufacturing enables to produce geometrically complex objects, shapes and textures. It often uses less material than traditional manufacturing methods and allows the production of prototypes / products that are not possible to produce economically with conventional manufacturing.

i) Familiarization of 3D printing machine and its principle of operation.
ii) Standard use of Computer Aided Design (CAD) drawings .dwg format and Interface with CURA / Simplify 3D software as .stl file.
iii) Selection of Polylactic Acid (PLA) and Acrylonitrile Butadiene Styrene (ABS) materials and their specifications.

1.2 Preparation of stepped pulley with PLA material as shown in Figure 1.1

- Slicing of stepped pulley using. stl format
- Laying of stepped pulley using 3D printing

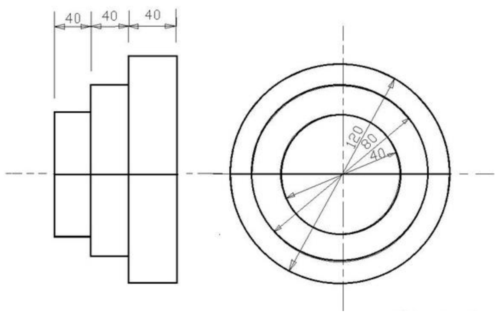


Figure 1.1 Stepped pulley

Try

1. Preparation of spur gear with ABS material as shown in the Figure 1.2
   - Slicing of spur gear using .stl format
   - Laying of spur gear using 3D printing.

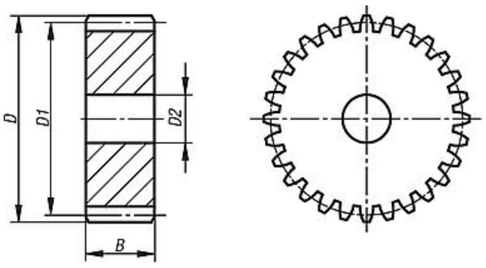


Figure 1.2 Spur gear

Dimensions:

D = 40mm

D1 = 44mm   Module, m= 2mm

D2= 20mm   B= 10mm

Hint: Set the appropriate nozzle and temperature for ABS material, compare with PLA material.

2. Introduction to computerized numerical control (CNC) laser engraving

CNC Engraving machine is the process of gradually removing a small amount of material from a surface along a defined path. The process leaves a visible marking on the surface of the substrate (workpiece) with high accuracy and precision control.

i) Familiarization of CNC engraving machine with action control.

ii) Create a visualization of the engraving pattern on CAD software. The student must have a solid grip on CAD to be able to create complex patterns in a quick and accurate manner.

iii) Computer Aided Manufacturing (CAM) software is a special software that is used to generate programs for the CNC engraving machine.

2.1 Preparation of acrylic gears using CNC laser engraving / cutting machine as shown in Figure 2.1

i) The parts file is divided into two pages. The file is a pdf. As different laser cutters use different file formats I leave it to you to convert it to a format suitable for your machine.
ii) All the parts apart for the dowel pieces cut out and ready to go. The blue piece is the clear acrylic front with its protective film still attached. Having cut out all the parts follow the instructions below to assemble the gear display.
iii) Glue together the two eighteen teeth gears. Make sure that they are aligned precisely.

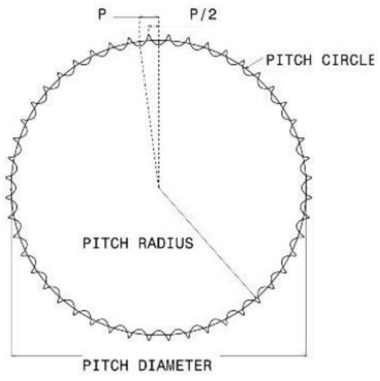


Figure 2.1 Acrylic gear

Try

1. Preparation of artistic components IARE logo using CNC laser engraving as shown in the Figure 2.2

- Open your program and create new file (OPEN-NEW FILE), use the TEXT tool and write the word you want to engrave.
- Shows how to transform this word into an objects, each letter will become an individual shape.
- When using the laser technology, you have to make sure the file is one whole object, not a group of objects.

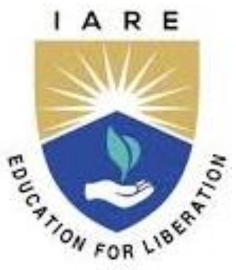


Figure 2.2 IARE logo

Hint: Set the appropriate parameters to perform operation on the balsaw wood material.

3. Introduction to computerized numerical control wood carving machines

CNC wood carving is the process of gradually removing a small amount of material from a surface along a defined path. The process leaves a visible marking on the surface of the substrate (workpiece) with high accuracy and precision control.

i) Familiarization of CNC machine with action control.
ii) Create a visualization of the engraving pattern on CAD software. The student must have a solid grip on CAD to be able to create complex patterns in a quick and accurate manner.
iii) Computer Aided Manufacturing (CAM) software is a special software that is used to generate programs for the CNC engraving machine.

3.1 Preparation of wooden wheel using computerized wood carving machine as shown in the Figure 3.1

- Preparation of CAD .dwg file
- Importing the file into wood carving machine for generating the profile using CAM software.

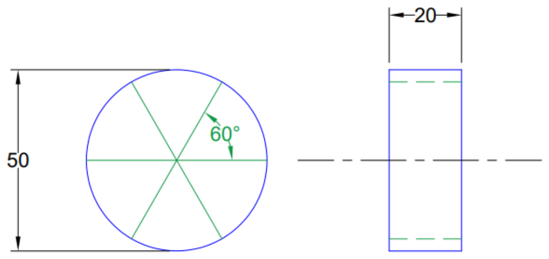


Figure 3.1 Wooden wheel

Try

1. Preparation of IARE lettering using CNC wood carving as shown in the Figure 3.2.

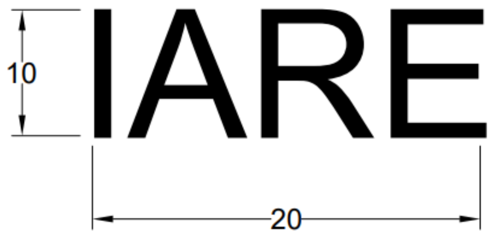


Figure 3.2 IARE lettering

Hint: Set the appropriate parameters to perform operation on the medium density fibreboard (MDF) wood material.

4. Introduction to pipe fitting and threading

Threaded fittings are used in non-critical applications and when service fluid is at ambient temperatures such was instrument air, plant air, cooling water, potable water etc. As they do not require welding, they are used at places where welding is not permitted.

i) Preparation of Polyvinyl Chloride (PVC) material for pipe fitting

ii) Making threads on PVC pipe using thread die sets

iii) Fitting the threaded PVC pipe for T-shape using Tee joint

4.1 Preparation of PVC material for pipe threading and fitting as shown in the figure 4.1

i) Start by cutting the two pipes to the required length, making sure that they are the same size and the ends line up properly.

ii) Make the threads using die sets to one end of the pipes, then place it on top of the pipe where you want to join.

iii) Then start rotating to get bonded together.

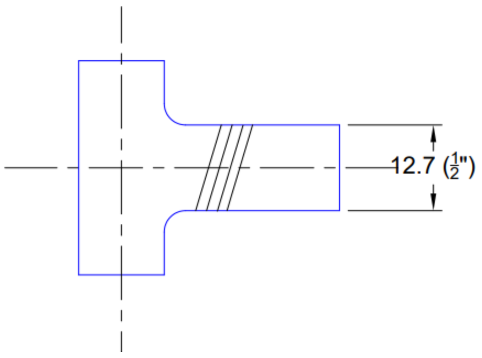


Figure 4.1 T Joint

Try

1. Preparation of galvanized steel I joint as shown in the Figure 4.2.

i) Start by cutting the two pipes to the required length, making sure that they are the same size and the ends line up properly.
ii) Make the threads using die sets to one end of the pipes, then place it on top of the pipe where you want to join.
iii) Then start rotating to get bonded together.

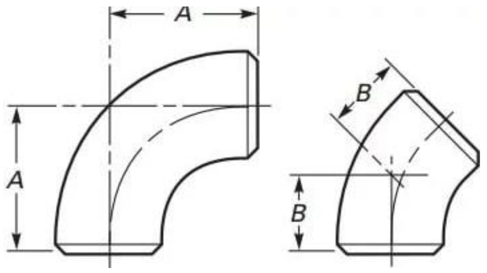


Figure 4.2. GI Elbow Joint

Dimensions:

A = 40mm

B = 30mm

Hint: Set the appropriate die sets in order to get perfect threading in pipe fittings.

5. Introduction to computer numerical control (CNC) lathe machines

A lathe is a machine tool that rotates a workpiece about an axis of rotation to perform various operations such as cutting, sanding, knurling, drilling, deformation, facing, and turning, with tools that are applied to the workpiece to create an object with symmetry about that axis.

i) Operate with Computer Numerical Control (CNC) systems and provided with precise design instructions.
ii) CNC Lathes are machine tools where the material or part is clamped and rotated by the main spindle, while the cutting tool that work on the material, is mounted and moved in various axis.

5.1 Preparation of Mild Steel (MS) material for step turning with grooving operation as shown in the Figure 5.1

- Inspect the mild steel raw material using Vernier calipers. The work piece is held in the chuck by placing it properly and tightening it using the chuck key.
- Now single point cutting tool is placed in the tool post and properly arranged to the centre of the work piece.
- Work piece is rotated by switching on the motor.
- Perform the facing operations on both sides and maintain the given dimensions.
- First the plain turning operation is carried out by placing the tool at 1 mm feed to the lathe axis.
- After that step turning operation is performed till the desired diameter is obtained.

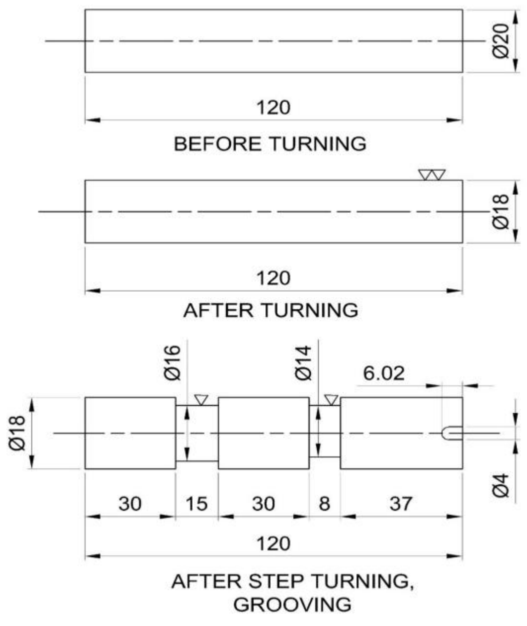


Figure 5.1. Step turning with grooving operation

Try

1. Preparation of Mild Steel (MS) material for step turning with tapper operation as shown in the Figure 5.2.
   - Select a tool bit to the desired size and shape of the groove required.
   - Lay out the location of the groove.
   - Set the lathe to half the speed for turning.
   - Mount the workpiece in the lathe and set the tool bit to centre height.

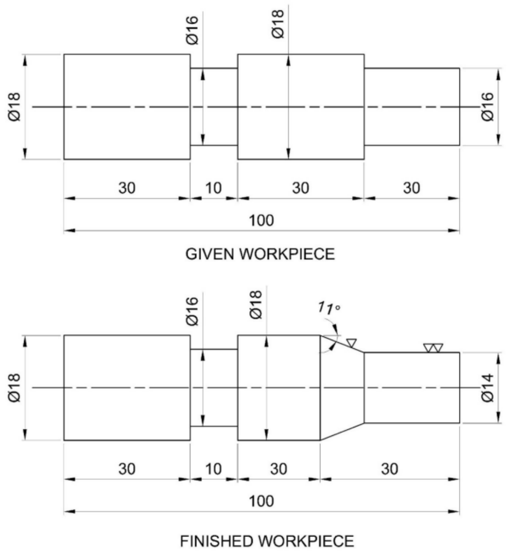


Figure 5.2. Step turning with tapper operation

6. Introduction to conventional lathe machines

The conventional lathe machine is a standard lathe that is used for holding and turning various types of materials such as metal, wood, plastic etc. against a cutting tool in order to produce a cylindrical object. Besides this it can even perform many other functions like grinding, boring, threading, polishing, reaming, drilling etc.

i) Operate with conventional lathe provides with manual design instructions.
ii) Lathes are machine tools where the material or part is clamped and rotated by the main spindle, while the cutting tool that work on the material, is mounted and moved in various axis.

6.1 Preparation of mild steel (MS) material for thread cutting and knurling operation as shown in the Figure 6.1

- Fix the job on the machine by using chuck key. Turn the job to the required diameter by fixing the single point cutting tool.
- Chamfer the edge and make an under cut at the other end.
- Engage the bed screw and perform the threading operation.
- Stop when the pitch is measured by the pitch gauge.
- Reverse the job and hold it carefully so that the threads are not damaged. Disengage the back gear and lead screw
- Hold the knurling tool against the rotating job.

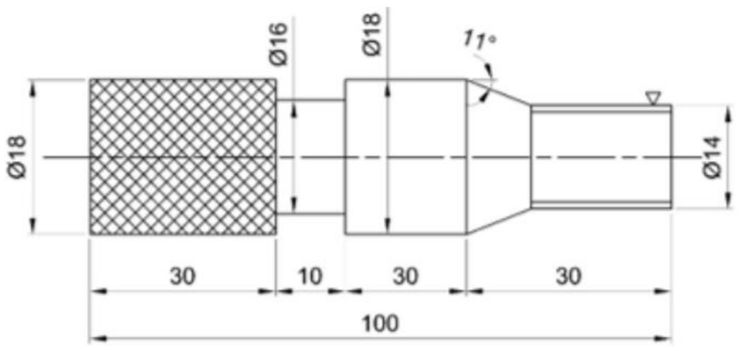


Figure 6.1. Thread cutting and knurling operation

Try

1. Preparation of aluminum material for step turning with tapper operation as shown in the Figure 6.2.
   - Select a tool bit to the desired size and shape of the groove required.
   - Lay out the location of the groove.
   - Set the lathe to half the speed for turning.
   - Mount the workpiece in the lathe and set the tool bit to centre height.

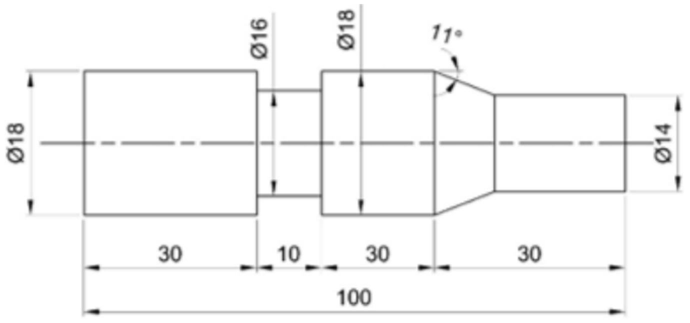


Figure 6.2. Step turning with tapper operation

7. Introduction to milling machines

Milling is the process of machining using rotary cutters to remove material by advancing a cutter into a workpiece. This may be done by varying directions on one or several axes, cutter head speed, and pressure. Milling covers a wide variety of different operations and machines, on scales from small individual parts to large, heavy-duty gang milling operations. It is one of the most used processes for machining custom parts to precise tolerances.

i) Milling machine employed in the metal removing operation in which the work is rigidly clamped on the table of the machine and the revolving cutter which has multiple teeth is mounted on the arbor.

ii) The cutter revolves at high speed and the work is fed slowly past the cutter.

iii) The work can be fed in a vertical, longitudinal, or cross direction depending upon the type of milling machine being used.

iv) As the work proceeds, the cutter-teeth removes the metal from the surface of the job(workpiece) to produce the desired shape.

7.1 Preparation of slotting operation as shown in Figure 7.1

- Keep the work piece on the working table in req. position with the help of holding device.
- Keep the cutting tool in the spindle and move the working table upward to give touch the surface of the work piece.
- Then give the power supply and move the work table forward and backward with the help of lever.
- Repeat the same procedure by changing the feed rate in upward and cross direction to get the req. dimension of slot on the work piece.

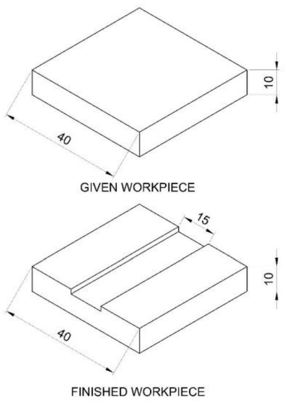


Figure 7.1 Slotting operation

Try

1. Perform the boring and reaming operation on a rectangular work piece to obtain the required dimensions using vertical milling machine as shown in Figure 7.2.

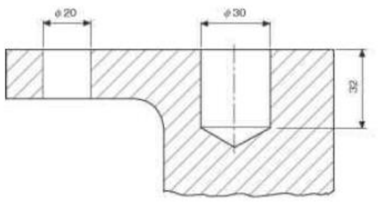


Figure 7.2. Boring and reaming operation

8. Introduction to shaping machines

A shaping machine is a mechanical device used to shape and form metal workpieces. It operates by removing material through a reciprocating cutting motion, resulting in the desired shape or contour. Shaping machines are commonly used in metalworking industries for various applications, including creating flat surfaces, slots, and grooves.

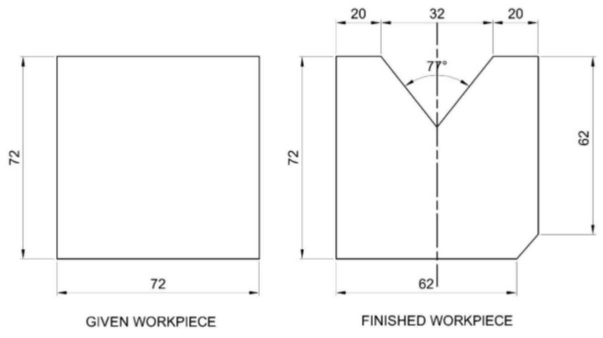
i) A rigid table on the machine supports the workpiece. Over the workpiece, the ram moves back and forth as shown in the animation above. A vertical tool slide is adjusted to either side of the vertical plane along the stroke axis, which is located at the front of the ram.

ii) The geometry of the linkage causes the ram to travel more quickly on the return stroke than the forward stroke (cutting stroke). As the shaper works on the quick return mechanism, the sliding action of the slider is aided by the rotating link.

iii) One of the four mechanisms i.e. crank and slotted, whitworth quick return, hydraulic, and automatic table feed mechanism, is responsible for the quick return mechanism and reciprocating movement of the ram. The automatic table feed is commonly used today which employs a pawl and ratchet mechanism in a shaping machine.

8.1 Preparation of V-groove operation as shown in Figure 8.1

- The job is fixed on a vice and the tool is fixed on tool post.
- The stroke of ram is adjusted to required length and machine is switched on.
- Always during machining, the job should be properly fixed with the half of try Square and vice to get a right-angle surface after machining.
- After completion of work, the job should be filled help of file before fixing the job, V block dimensions are marked on the job with the help of dot punch.
- The tool head should be rotated at 450 to make the V- groove.
- The feed is given such that the tool moves gradually on either side of the middle line.



•

Figure 8.1 V-groove operation

Try

1. Perform the key ways on a cylindrical work piece to obtain the required dimensions using shaping machine as shown in Figure 8.2

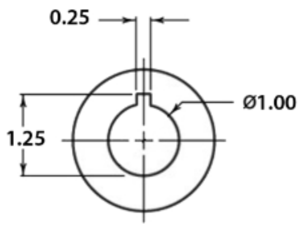


Figure 8.2 Key ways on a cylindrical work piece

9. Introduction to electrical wiring practices for domestic appliances

The Electrical Wiring Systems are mostly standardized with several rules, regulations, and laws. Electrical Wiring must be installed correctly and safely in accordance with electrical regulations and standards. If the electrical wiring is carried out incorrectly or without confirming to any standard, then it may lead to incidents like short circuits, electric shocks, damage the device / appliance or leads to the malfunctioning of device which further causes for the reduction of device life.

i) Before starting any installation work, the first and foremost thing is the concern of safety of the personnel. Electricity is dangerous and direct or indirect contact of electrical equipment or wires with the power turned ON can result in serious injuries or sometimes even causes death. Follow the below steps to maintain the safety at the workplace.

ii) Several factors must be considered before the actual installation work to be done for residential, commercial, or industrial wiring. These factors include type of building construction, type of ceiling, wall, and floor construction, wiring methods, installation requirements, etc.

9.1 Preparation of wiring for a stair case arrangement using a two-way switch as shown in Figure 9.1

- Mark switch and bulb location points and draw lines for wiring on the wooden Board.
- Place wires along the lines and fix them with the help of clips.
- Fix the two-way switches and bulb holder in the marked position on the wooden Board.
- Complete the wiring as per the wiring diagram.
- Test the working of the bulbs by giving electric supply to the circuit

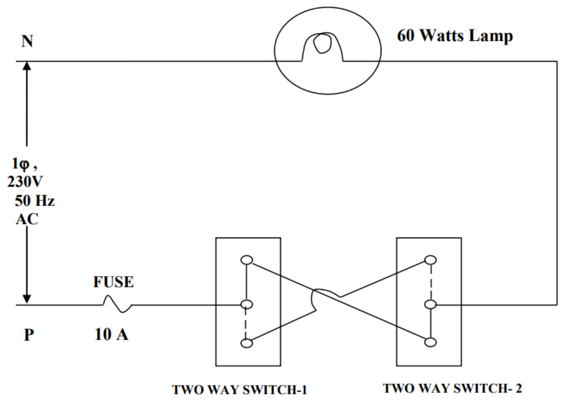


Figure 9.1 Circuit diagram-staircase wiring

Try

1. Prepare wiring for a tube light with switch control as shown in Figure 9.2.
   - Mark the switch and tube light location points and draw lines for wiring on the wooden board.
   - Place wires along the lines and fix them with the help of clips.
   - Fix the switch and tube light fitting in the marked positions.
   - Complete the wiring as per the wiring diagram.
   - Test the working of the tube light by giving electric supply to the Circuit.

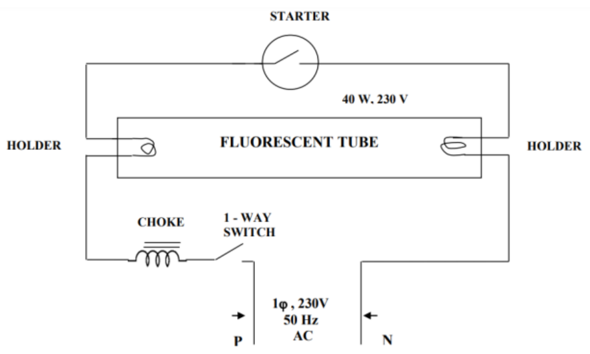


Figure 9.2 Circuit diagram-tube light

10. Introduction to soldering and desoldering practice

Soldering is defined as "the joining of metals by a fusion of alloys, which have relatively low melting points". In other words, you use a metal that has a low melting point to adhere the surfaces to be soldered together. Soldering is more like gluing with molten metal than anything else. Soldering is also a must have skill for all sorts of electrical and electronics work. It is also a skill that can only be developed with practice.

i) Soldering requires two main things: a soldering iron and solder. Soldering irons are the heat source used to melt solder. Irons of the 15W to 30W range are good for most electronics/printed circuit board work.
ii) Using anything higher in wattage and you risk damaging either the component or the board. Note that you should not use so-called soldering guns. These are very high wattage and generate most of their heat by passing an electrical current through a wire. Because of this, the wire carries a stray voltage that could damage circuits and components. The choice of solder is also important.
iii) One of the things to remember is to never use acid core solder. Acid core solder will corrode component leads, board traces and form conductive paths between components.

10.1 Preparation of soldering from a circuit board as shown in Figure 10.1

- All parts must be clean and free from dirt and grease.
- Try to secure the work firmly.
- "Tin" the iron tip with a small amount of solder. Do this immediately, with new tips being used for the first time.
- Clean the tip of the hot soldering iron on a damp sponge.
- Many people then add a tiny amount of fresh solder to the cleansed tip.
- Heat all parts of the joint with the iron for under a second or so.
- Continue heating, then apply sufficient solder only, to form an adequate joint.
- Remove and return the iron safely to its stand.
- It only takes two or three seconds at most, to solder the average printed circuit board (PCB). joint.
- Do not move parts until the solder has cooled.

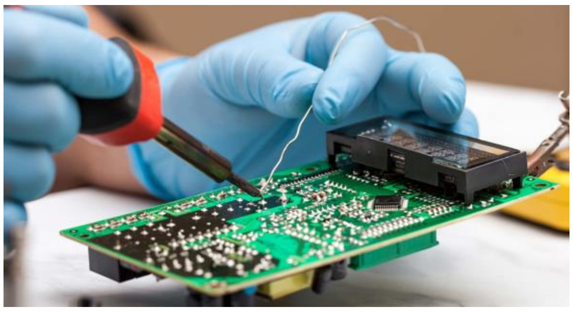


Figure 10.1 Soldering operation

Try

1. Perform desoldering operation from a circuit board as shown in Figure 10.2.
   - Heat up the solder with the iron.
   - Slide the iron up the pins to bring most of the solder away from the joint.

- Using pliers, gently pull at the components to remove their pins from the pin holes while they are still hot. It's a good idea to pull by their leads as opposed to on the components themselves to maintain the quality of the component.

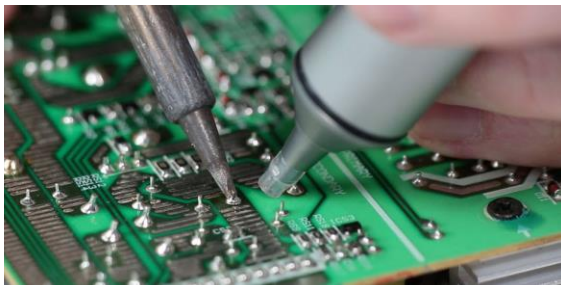


Figure 10.2 De-Soldering operation

11. Introduction to troubleshooting the ceiling fan and mixer grinder

In generally most ceiling fan and table fan have a capacitor start permanent capacitor motors which a difference from usual motor that it central position (Rotor/ Armature) remains fixed, while the outer portion rotates blades are mounted on the outer shaft when the motor is energized the blade cause to rotate and to circulate the surroundings are depends on the speed of fan. A regulator is connected in series with fan at different tapings hence the speed of the fan consist of a number of parts are connected together as a shaft to avoid loose fittings the parts are located bolts, split, pins and bearing lock.

i) Usually, a ceiling fan and table fan consist of a capacitor start and run motor. Capacitor are always connected in the circuit the advantages of leaving the capacitor permanently in circuit.

ii) It has one starting winding in series with one capacitor and running winding since the capacitor remains in the circuit permanently. This motor is often referred to as permanent. Split capacitor runs motor and behaves practically like an unbalanced 2phase motor.

11.1 Perform the maintenance of ceiling fan and ending the trouble shoot problems as shown in Figure 11.1

- The rapid spinning and vibrations your fan’s components are subjected to can cause them to work loose and wear out. Inspecting the fan every two or three months you use it helps keep the fan working efficiently and extends its lifespan.
- One of the most common problems is a loose mount, which can cause your fan to wobble. A wobbling fan is unlikely to fall, but it can cause the light fixture to fall, so it’s not something to ignore.
- Blades that are misaligned or out of balance can also cause wobbling. Over time, one or more blades can become warped, bent or otherwise damaged. Even a minor difference can disrupt the fan’s normal performance, so check each blade closely.

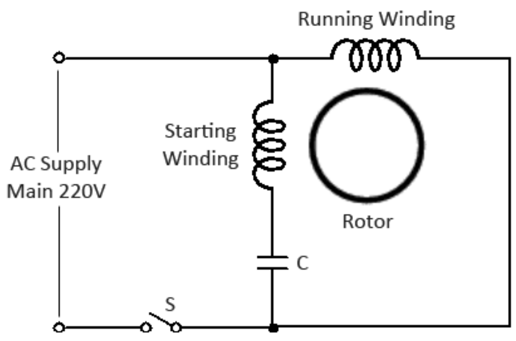


Figure 11.1 Celling fan circuit diagram

Try;

1. Perform the maintenance for mixer grinder from a circuit board as shown in Figure 11.2.
   - Universal Series Motor
   - Three Position Speed Control Rotary Switch
   - Thermal Overload Relay or overload switch
   - Indicator Light

- Power Switch

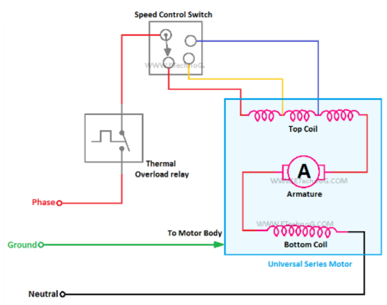


Figure 11.2. Mixer grinder circuit diagram

12. Introduction to 6 axis articulated robotic arm

ARISTO is six axis articulated robotic arm of industrial for training and research and is manufactured to industrial standards. The robot is capable of lifting up to 2.5kg of pay load. The robot can be used with pneumatic or electrical grippers. ARISTO has simulation software that allows the user to learn robot functions, application & programming.

- The evolution of the performance of robots and programming software provides new machining solutions. For complex parts, six axes robots offer more accessibility than a machining center CNC 5 axis and allow the integration of additional axes to extend the workspace.
- Robots have seen in recent years an expansion of their field of use with new requirements related to the increasing use of composites. The robots are then considered for machining operations (polishing, cutting, drilling etc.) that require high performance in terms of position, orientation, followed by trajectory precision and stiffness.

12.1 Preparation of articulated robot for lifting load as shown in Figure 12.1

- A load lifting robot is a type of industrial robot that is used for handling and placing products on a production line.
- They are typically used in high-volume manufacturing settings, where they can quickly and accurately place products onto conveyor belts or other production equipment

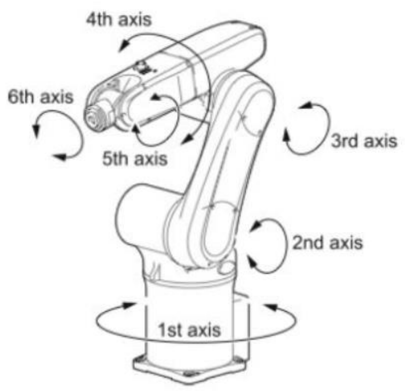


Figure 12.1 Six Axis Aristo Robot

Try

1. Perform the pick and place operation for the articulated robot as shown in Figure 12.2
   - A pick-and-place robot is a type of industrial robot that is used for handling and placing products on a production line.
   - They are typically used in high-volume manufacturing settings, where they can quickly and accurately place products onto conveyor belts or other production equipment.

Figure 12.2. Pick and place operation

13. Introduction to FANUC simulator

The FANUC CNC Simulator brings the world's most popular control right into your workplace training room, providing hands-on training for FANUC CNC operation without the need for a full machine. Add Machining Simulation Software to the CNC Simulator for advanced machine simulation capability.

- This PC-based platform is perfect for training and designing part programming. The CNC Machining Simulation software provides a digital twin of the machine tool producing the real-world cutting process. This provides you with the most realistic simulation of the actual machining on your floor. To prepare the industry for more complex machining knowhow, a 5-axis machining training option is now available.
- For companies needing a more tailored training offering, FANUC America's CNC Hardware Simulators are fully functional CNCs that include the panel and operating system. More robust than our CNC Simulators, our CNC Hardware Simulators are complete control simulators customized to address your specific workplace training needs. Operations needing a particular CNC model for their shopfloor training will benefit from these simulators.

13.1 Preparation of milling and lathe system switchable on one simulator as shown in Figure 13.1

- Switchable mill and lathe (turning) system in one simulator
- 3-axis milling / 2-axis turning system plus one spindle
- Manual Guide is installed for conversational program creation and 3D simulation
- Imperial / metric switchable
- 512KB part-program storage, with 400 registered programs
- 32 tool offset pairs
- Workpiece coordinates G52 – G59 plus 48 additional on mill

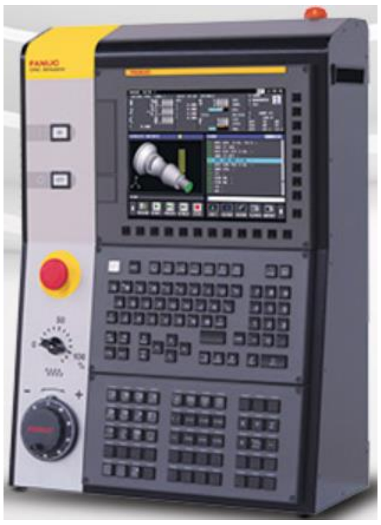


Figure 13.1 FANUC Simulator

Try

1. Perform the combination of CNC Simulator with CNC machining simulation software as shown in Figure 13.2.
   - The CNC Machining Simulator carries all the same features as the standard simulator with the addition of an internal PC, which will provide a virtual mill or lathe with real kinematics, allowing the user to view the live machine movement, tooling and part machining.
   - Tool and workpiece setup is required and colored back plot, collision detection and sound bring the machine operating experience to life.

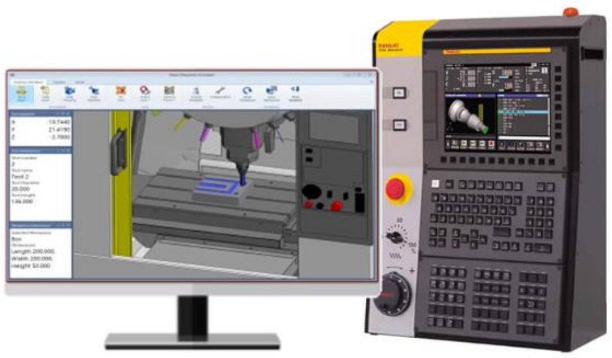


Figure 13.2 CNC Simulator with CNC machining simulation

14. Demonstration to cylindrical grinding machine

Most commonly, cylindrical grinding is used for grinding pieces with a central axis of rotation, like rods and cylinders. This process involves using a cylindrical grinder, which is a type of machinery categorized by rotation style and wheel device.

- A grinding machine uses an abrasive product usually a rotating wheel to shape and finish a workpiece by removing metal and generating a surface within a given tolerance. A grinding wheel is made with abrasive grains bonded together. Each grain acts as a cutting tool, removing tiny chips from the workpiece.

14.1 Demonstration on industry standard grinding as shown in Figure 14.1

- The ECO-200 Micromatic industry standard universal cylindrical grinding machine is used for grinding of components up to an accuracy of 5m and used for projects.

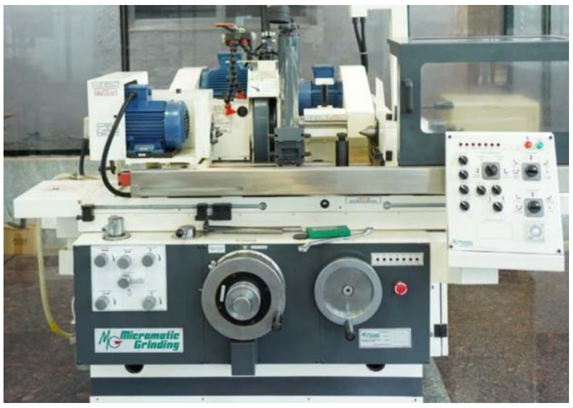


Figure 14.1 Cylindrical griding

Try

1. Demonstration grinding methods and machines
   - Grinding, or abrasive machining, once performed on conventional milling machines, lathes and shapers, is now performed on various types of grinding machines.
   - Grinding machines have advanced in design, construction, rigidity and application far more in the last decade than any other standard machine tool in the manufacturing industry. Grinding machines fall into five categories: surface grinders, cylindrical grinders, centerless grinders, internal grinders and specials.

V. TEXT BOOKS:

1. Hajra Choudhury S.K., Hajra Choudhury A.K. and NirjharRoy S.K., *Elements of Workshop Technology*, Media promoters and publishers private limited, Mumbai, 2020.
2. Kalpakjian S, Steven S. Schmid, *Manufacturing Engineering and Technology*, Pearson Education India Edition, 7th Edition, 2019.

VI. REFERENCE BOOKS:

1. Gowri P. Hariharan, A. Suresh Babu, *Manufacturing Technology – I*, Pearson Education, 2018.
2. Roy A. Lindberg, *Processes and Materials of Manufacture*, Prentice Hall India, 4th Edition, 2017.
3. P.N., *Manufacturing Technology*, Vol. I and Vol. II, Tata McGraw-Hill House, 2017.
4. Rupinder Singh, J. Paulo Davim, *Additive Manufacturing: Applications and Innovations*, CRC Press, 2nd Edition, August, 2021.
5. Jeyaprakash Natarajan, Muralimohan Cheepu, Che-Hua Yang, *Advances in Additive Manufacturing Processes*, Bentham Books, 4th Edition, September, 2021.

VII. ELECTRONICS RESOURCES:

1. https://elearn.nptel.ac.in/shop/iit-workshops/ongoing/additive-manufacturing-technologies-for-practicing-engineers/.
2. https://akanksha.iare.ac.in/index?route=course/details&course_id=94.

VIII. MATERIALS ONLINE:

1. Course Template
2. Lab manual

COURSE CONTENT

ELECTRICAL AND ELECTRONICS ENGINEERING LABORATORY								
I Semester: CSE (AI&ML) / IT / AERO / MECH / CIVIL **II Semester:** CSE / CSE(DS) / CSE(CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AEED03	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Nil								

I. COURSE OVERVIEW:

This course **serves** as a foundation course on electrical engineering. It **covers** a broad range of fundamental electrical circuits and devices. The **concepts** of current, voltage, power, basic circuit elements, electrical and electronic devices and their **application** in more complex electrical systems are to be imparted to the students.

II. COURSES OBJECTIVES:

The students will try to learn:

I. The basic laws for different circuits.
II. The elementary experimental and modeling skills for handling problems with electrical machines in the industries and domestic applications to excel in professional career.
III. The intuitive knowledge needed to test and test and analyze the performance leading to design of electric machines by conducting various tests and calculate the performance parameters.
IV. The semiconductor devices like diode and transistor.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Solve an electric circuit by providing laws and solving theorems.
CO2	Analyze the performance characteristics of DC shunt machine at various loading conditions
CO3	Examine the performance of induction motors by conducting a suitable test.
CO4	Acquire basic knowledge on the working of diodes to plot their characteristics
CO5	Identify transistor configuration and their working to deduce its working.
CO6	Use of the two port parameters to be measured easily, without solving for all the interna voltages and currents in the different networks.

Dos

1) For safety purpose the students should compulsory wear leather shoes.
2) Students should come in uniform prescribed.
   i. For boys, half sleeve shirts, tucked in trousers
   ii. For ladies, half sleeve overcoat, hair put inside the overcoat
3) After giving connections, staff members should be asked to verify the circuit connections.
4) Before staring the circuit connections check whether the circuit breaker is in OFF condition.
5) Circuit should be switched ON only after getting permission from the staff member.
6) To be careful with moving parts in the machine.
7) To come prepared with procedure relevant to the experiment.
8) Unplug electrical equipment after use.

Dont's

1) Don't assume that the power is disconnected.
2) Don't attempt to repair electrical equipment.
3) Don't come with any ornaments when working with electrical machines.
4) Don't use an earth connection as a neutral.
5) Don't touch any parts unnecessarily.
6) Don't keep any fluids and chemicals nearing instruments and circuits.

IV. COURSE CONTENT:

EXERCISES FOR ELECTRICAL AND ELECTRICAL ENGINEERING LABORATORY

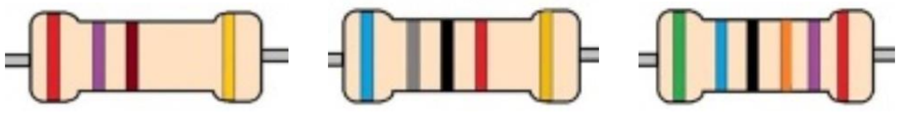
Note: Students are encouraged to bring their own laptops for laboratory practice session

1. Getting Started Exercises

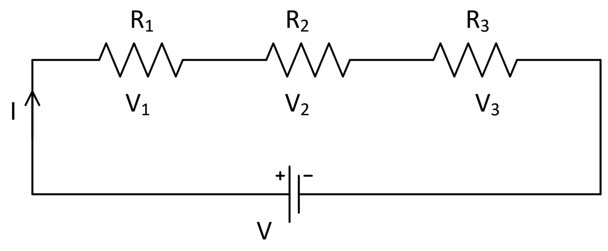
1.1 Introduction to electrical circuits

1. Understand the basic electrical equipment's used in the laboratory.
2. Become familiar with the operation and usage of basic DC electrical laboratory devices, namely DC power supplies and digital multimeter's.
3. Learn the measurement of resistance values using colour code and digital multimeter.
4. Learn the basics of circuit design using Simulink.

Try
1.Calculate the resistance value of Resistor – 1, Resistor – 2 and Resistor – 3 using colour code and verify using a digital multimeter.

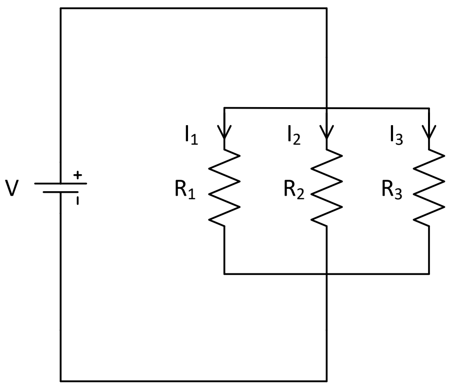


2. Design Circuit – 1 using Simulink and find the voltages V_1, V_2, V_3 and Current I. Where V_s = 6 V, R_1 = 100 Ω, R_2 = 220 Ω, R_3 = 1k Ω.



Circuit – 1

3. Design Circuit – 2 using Simulink and find the currents I_1, I_2 and I_3. Where V_s = 6 V, R_1 = 100 Ω, R_2 = 220 Ω, R_3 = 1k Ω.

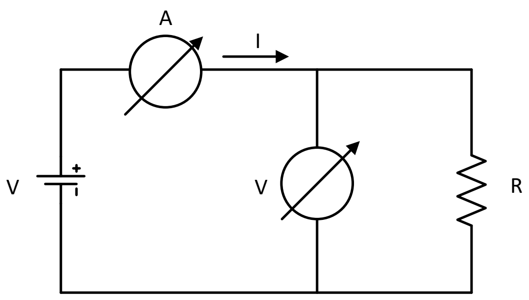


Circuit – 2

2. Exercises on Basic Electrical Circuit Law’s

2.1 Ohm’s law

1.Examine Ohm’s law of Circuit – 3 and draw the V-I characteristic of linear resistors R = 1k Ω



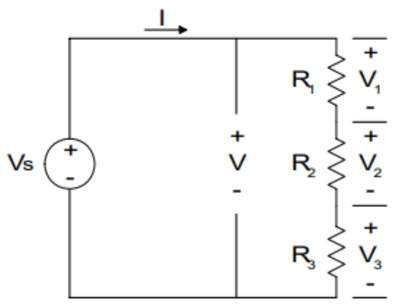
Circuit - 3

Try

1. Verify Ohm’s law of Circuit – 3 using Simulink and draw the V-I characteristics of a linear resistor R = 470 k Ω
2. An electric heater takes 1.48 kW from a voltage source of 220 V. Find the resistance of the heater.

2.2 Kirchhoff’s voltage law

1. Examine Kirchhoff’s voltage law using basic series DC Circuit - 4 with resistors. Where V_s = 6 V, R_1 = 100 Ω, R_2 = 220 Ω, R_3 = 1k Ω.



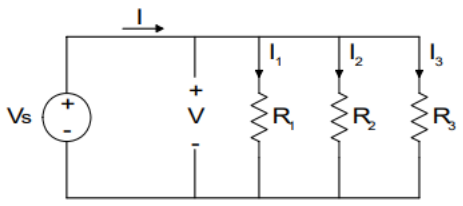
Circuit – 4

Try

1. Design and Verify Kirchhoff’s voltage law for Circuit – 4 using Simulink.
2. Determine the voltage V_2 by replacing the resistor R_2 = 150 Ω.
3. Find the total current I flowing through the Circuit – 4.

2.3 Kirchhoff’s current law

1. Examine Kirchhoff’s current law using basic parallel DC Circuits - 5 with resistors. Where V_s = 6 V, R_1 = 100 Ω, R_2 = 220 Ω, R_3 = 1k Ω



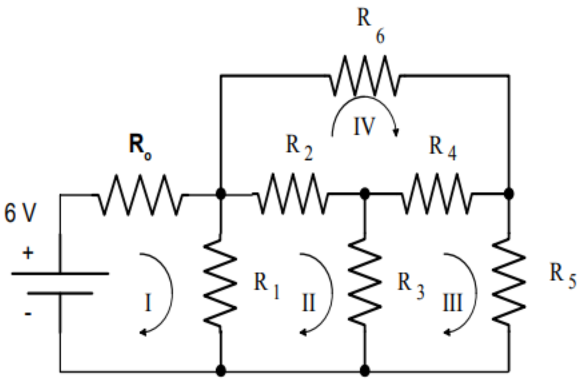
Circuit – 5

Try

1. Design and Verify Kirchhoff's current law for Circuit – 5 using Simulink.
2. Determine the voltage I_2 by replacing the resistor $R_2 = 150\ \Omega$.

3. Exercises on Mesh Analysis

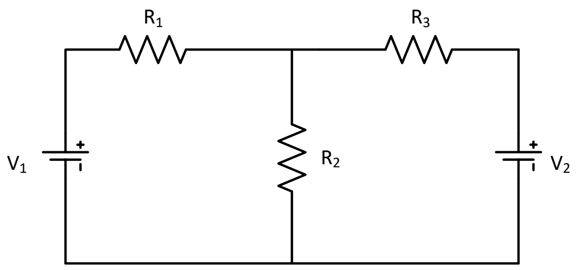
1.Determine mesh currents in the complex electrical Circuit - 6 by using principles of basic electrical circuits. Where $R_0 = 47\Omega$, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$, $R_3 = 1k\ \Omega$, $R_4 = 150\ \Omega$, $R_5 = 82\ \Omega$, $R_6 = 100\ \Omega$.



Circuit – 6

Try

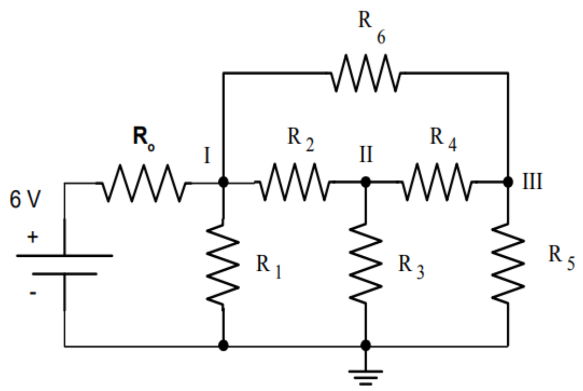
1. Use Simulink to simulate the Circuit – 6 for determining the current flowing through each resistor and compare these values to those you obtained from your experiments.
2. Find the current flowing through the resistor R_2 and R_3 by replacing the values of the resistors $R_2 = 100\ \Omega$, $R_3 = 470\ \Omega$.
3. Find the current flowing through the resistor R_3 for Circuit – 7 using mesh analysis when V1 = 10V, V_2 = 6V, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$ and $R_3 = 1k\ \Omega$.



Circuit – 7

4. Exercises on Nodal Analysis

1. Determine nodal voltages in complex electrical Circuit – 8 by using principles of basic electrical circuits. Where $R_0 = 47\Omega$, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$, $R_3 = 1k\ \Omega$, $R_4 = 150\ \Omega$, $R_5 = 82\ \Omega$, $R_6 = 100\ \Omega$.



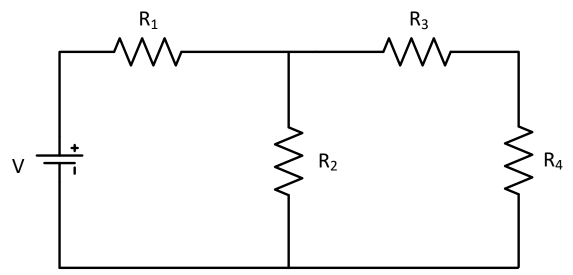
Circuit – 8

Try

1. Use Simulink to simulate the Circuit – 8 for determining the voltage across each resistor and compare these values to those you obtained from your experiments
2. Determine the voltage at node - II by replacing the resistor $R_2 = 150\ \Omega$
3. Find the voltage V_1 for Circuit – 7 using nodal analysis when V1 = 10V, $V_2 = 6V$, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$ and $R_3 = 1k\ \Omega$.

5. Exercises on Thevenin's Theorem

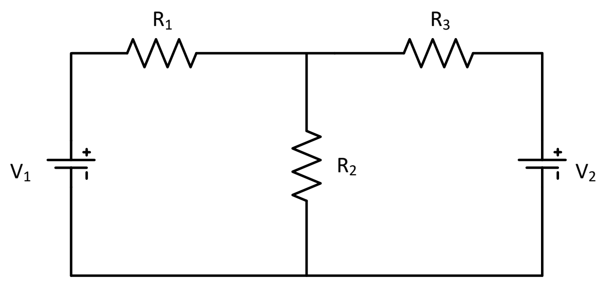
1. Determine Thevenin's equivalent voltage (V_{th}) and resistance (R_{th}) at the load terminals by applying Thevenin's theorem for Circuit – 9.
2. Determine load or unknown current through a R_4 resistor using Thevenin's equivalent circuit. Where V = 10V, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$, $R_3 = 1k\ \Omega$ and $R_4 = 150\ \Omega$



Circuit – 9

Try

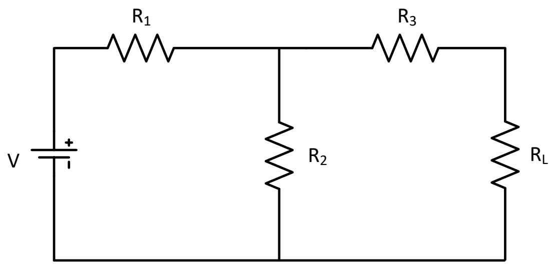
1. Use Simulink to simulate the Circuit – 13 for determining the current thought R_4 resistor using Thevenin's theorem and compare this value to those you obtained from your experiment.
2. Find the current thought R_4 resistor using any circuit reduction technique and verify this value to those you obtained from Thevenin's theorem.
   Find the current flowing through R_2 resistor in Circuit – 10 using Thevenin's theorem for the below circuit. Where $V_1 = 10V$, $V_2 = 5V$, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$, $R_3 = 1k\ \Omega$



Circuit – 10

6. Exercises on Norton's Theorem

1. Find Norton equivalent current (I_N) and resistance (R_N) by considering $R_L = 150\ \Omega$ resistor for the Circuit – 15 by applying Norton's theorem.
2. Find load or unknown current through R_L resistor using Norton's equivalent circuit. Where V = 10V, $R_1 = 100\ \Omega$, $R_2 = 220\ \Omega$ and $R_3 = 1k\ \Omega$.



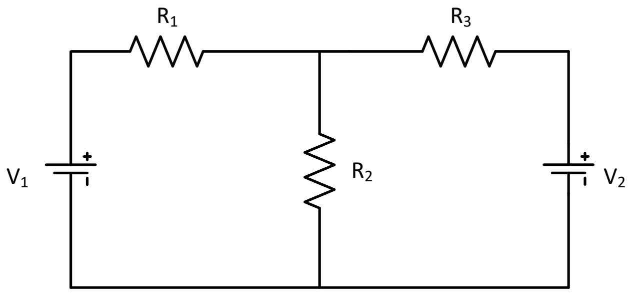
Circuit – 11

Try

1. Use Simulink to simulate the Circuit – 11 for determining the current through R_L resistor and compare this value to those you obtained from your experiment
2. Find the current thought R_L using any circuit reduction technique and verify this value to those you obtained from Norton's theorem.

7. Exercises on Superposition Theorem

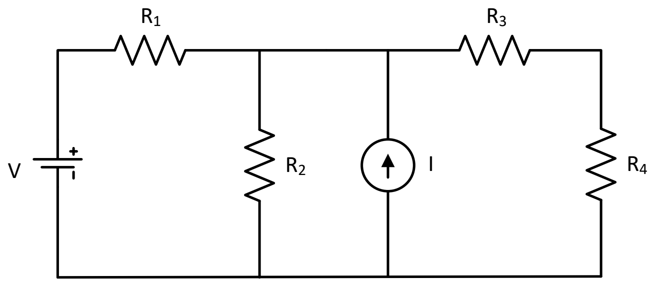
1. Investigate the current thought R_2 resistor using superposition theorem to multiple DC source Circuit - 16



Circuit – 12

Try

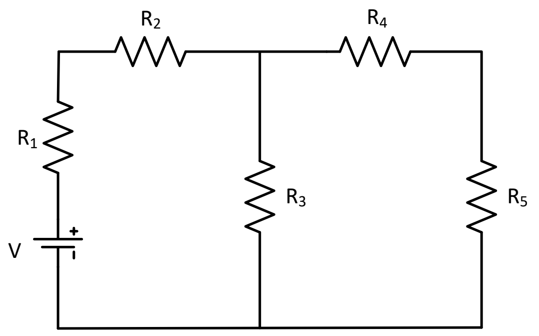
1. Use Simulink to simulate the Circuit – 12 for determining the current through a R_2 resistor and compare this value to those you obtained from your experiment
2. Find the current thought R_2 in Circuit – 12 using any circuit reduction technique and verify this value to those you obtained from the superposition theorem.
3. Find the current through the R_4 resistor using the superposition theorem for the Circuit – 13.



Circuit – 13

8. Exercises on Reciprocity Theorem

1. Understand the reciprocity theorem by analyzing Circuit – 14 with interconnected components using fundamental circuit laws where V = 10 V, R_1 = 100 Ω, R_2 = 220 Ω, R_3 = 1k Ω, R_4 = 150 Ω and R_5 = 82 Ω.



Circuit – 14

Try

1. Use Simulink to simulate Circuit–14 for determining the current through R_5 and R_1 by interchanging the voltage source in series with the R_5 resistor and compare this value to those you obtained from your experiment
2. Find the current thought R_3 using Thevenin's theorem.

9. Swinburne's test and speed control of dc shunt motor

1. Design the suitable test under no load conditions to measure no load losses in Dc shunt machines and speed control of DC shunt motor.

Try

1. Calculate the output power and efficiency when motor takes 10A on full load and 5A on half Load.
2. Measure the no load machine losses by using indirect method of testing.
3. Perform the speed control by varying the armature circuit resistance and field circuit resistance of DC shunt motor.

10 magnetization characteristics of dc shunt generator

1. Develop the circuit for analyzing the magnetization characteristics of DC shunt generator.

Try

1. From the Open circuit characteristics calculate the critical resistance of field winding.
2. Using magnetization characteristics calculate the critical speed of DC shunt generator at 100 Ω
3. Determine the performance of DC generator using the magnetization curve.
4. Calculate the critical value of shunt field resistance at 1500 rpm

11 Exercises on PN junction diode characteristics

Study the characteristics of PN junction diode as shown in Figure. 1

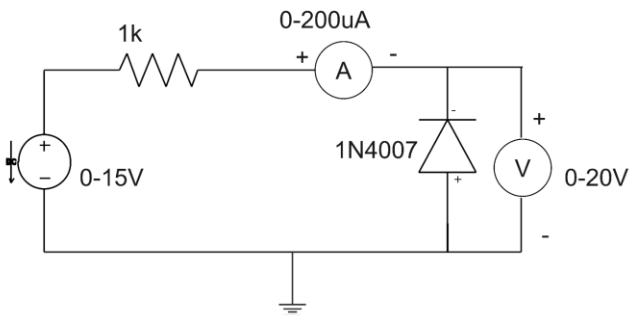


Figure. 1 diode as forward bias

Try

1. Plot the V-I Characteristics of germanium diode and find the cut in voltage.
2. Design diode acts as switch and plot the switching times of diode.

12 Zener diode characteristics and voltage regulator

Study the characteristics of Zener diode as shown in Figure. 2

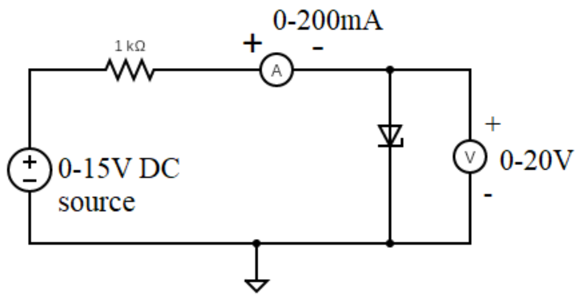


Figure. 2 Circuit diagram for Zener diode as forward bias

Try

1. Design a Zener voltage regulator circuit to drive a load of 6V, 100 mW from an unregulated
2. input supply of V_{min} = 8V, Vmax = 12V using a 6V Zener diode. .
3. Design square wave generator using Zener diode
4. Design for a Zener Transistor series voltage regulator circuit to drive a load of 6V, 1w,
5. from a supply of 10V with a ±3V ripple voltage

13 Half wave rectifier with/without filter

1. Design a half-wave rectifier circuit and analyze its output as shown in Figure. 3
2. Analyze the rectifier output using a capacitor in shunt as a filter as shown in Figure. 3

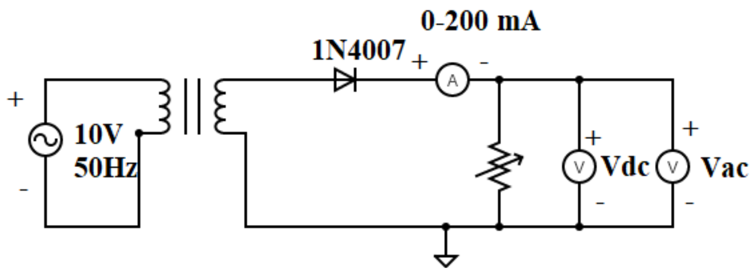


Figure. 3 Circuit Diagram for Half Wave Rectifier without Filter

Try

1. Design half wave rectifier with an applied input AC power is 100 watts, and it is to deliver an output power is 40 watts.
2. Design half wave rectifier with an AC supply of 230 V is applied through a transformer of turn ratio 10 : 1. Observe the output DC voltage, peak inverse voltage and identify dc output voltage if transformer turns ratio changed to 20:1.

14 Full wave rectifier with/without filter

1. Design a Full-wave rectifier circuit and analyse its output as shown in Figure. 4
2. Analyse the rectifier output using a capacitor in shunt as a filter as shown in Figure. 4

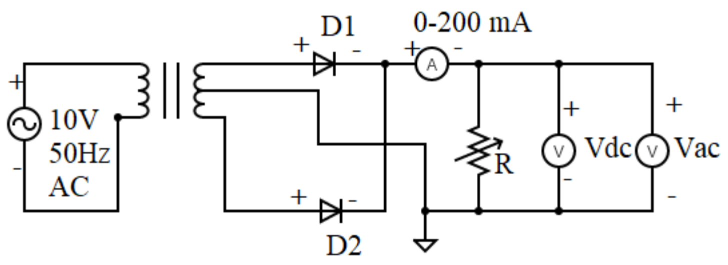


Figure 4: Circuit Diagram for Full Wave Rectifier without Filter

Try

1. Design a full wave rectifier with step down transformer and center tapped transformer. Justify the operation.
2. Design Full wave rectifier with capacitive filter using 10uF and 1uF. Observe the ripple

V. TEXT BOOKS:

1. A Chakrabarti, *CircuitTheory*, Dhanpat Rai Publications, 2004.

VI. REFERENCE BOOKS:

1. J P J Millman, C C Halkias, Satyabrata Jit, *Millman's Electronic Devices and Circuits*, Tata McGraw Hill, 2nd Edition, 1998.
2. RL Boylestad, Louis Nashelsky, *Electronic Devices and Circuits*, PEI/PHI, 9th Edition, 2006.

VII. ELECTRONICS RESOURCES:

1. https://www.nptel.ac.in/Courses/117106108
2. https://www.gnindia.dronacharya.info/EEEDept/labmanuals.html
3. https://www.textofvideo.nptel.iitm.ac.in
4. https://www.textofvideo.nptel.iitm.ac.in/

VIII. MATERIALS ONLINE

11. Course template
12. Lab manual

COURSE CONTENT

ESSENTIALS OF INNOVATION								
I Semester: AE / ME / CE / ECE / EEE / CSE (AI&ML) / IT **II Semester:** CSE / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD03	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	2	-	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: There are no prerequisites to take this course								

I. COURSE OVERVIEW:

Essentials of Innovation and Design thinking is a strategic approach towards creative problem-solving by placing users'/customers' needs above everything else. It is a process of questioning: questioning the problem, questioning assumptions, and questioning the implications. As a process it is a great catalyst of change and evolution. A Design thinking approach helps develop and build a culture of innovation across the students.

II. COURSE OBJECTIVESS:

I. The implications of disruption and the role of innovation.
II. The various frameworks, tools, and techniques of design thinking.
III. How to design, develop and implement an innovation product or service or process.

III. COURSE CONTENT:

Module-I; Philosophy of Innovation and Design Thinking

- Introduction to Innovation and Design Thinking
- History and Philosophy of Design Thinking
- Design Thinking as Problem-Solving Tool
- Design Thinking and it's Benefits
- Design Thinking Mind-set

Module-2: Mechanics of Innovation and Design Thinking

- Integrative View of Design Thinking
- Design Thinking Process
- 5 Stages (Empathise, Define, Ideate, Prototype and Test)
- Conceptual Frameworks Used in Design Thinking Process
- Case Studeis

Module-3: Design Thinking for Understanding Customers

- Understanding the User and Context
- Market Research
- Visualization and Customer Journey Mapping
- Empathy Mapping
- Redefining Problems, Brainstorming
- Reframing the Perspectives
- Ideation and Creativity
- Creative Ideation Methodologies
- Sketching & Visualization
- Storytelling

Module-4: Implementing Design Thinking

- Innovating Products, Services and Business Models
- Concept Evaluation and Concept Development
- Applications of Design Thinking
- Designing for Tangibles and Intangibles
- Ideas and Opportunities for Products

Module-5: Innovation Management

- Introduction to Innovation Management
- Business, Product & Process Innovation
- Organization Innovation
- Innovating Products, Services and Business Models
- Crafting a Better World Using Design Thinking & Innovation
- Design Thinking, Innovation and Organization Strategy
- Idea Pitching and Validation

Text Books:

I. Nigel Cross, "*Design Thinking: Understanding How Designers Think and Work*", Kindle Edition, 2011.
II. Tim Brown, Harper Bollins, "*Change by Design*", 2009.
III. Idris Mootee, "*Design Thinking for Strategic Innovation*", John Wiley & Sons, 2013.

COURSE CONTENT

ENVIRONMENTAL SCIENCE								
I Semester: AE / ME / CE / ECE / EEE/ CSE (AI & ML) / CSE / CSE (CS) / CSE (DS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSD06	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	-	-	-	-	-
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: Nil**		
Prerequisite: Basic Principles of earth science.								

I. COURSE OVERVIEW:

This course is an interdisciplinary study which examines the interaction between humans and the environment, with specific reference to the effects of modern technological advances. The students will be able to understand the sustainable development, ecological sustainability, environmental pollution, environmental issues in order to protect the environment and followed by the application of this knowledge to current environmental problems in the later years.

II. COURSES OBJECTIVES:

The students will try to learn

I. The interrelationship between living organism and environment.
II. The importance of environment by assessing its impact on the human world
III. The knowledge on themes of biodiversity, natural resources, pollution control and waste management.
IV. The sustainability and unsustainability of various interactions between human society and the earth's natural systems

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Infer the basic ecological principles, biogeochemical cycles and its function for the flow of energy in ecosystem.
CO2	Awareness on different natural resources and its conservation for sustainable development.
CO3	Use alternate energy resources for future growing energy needs.
CO4	Predict the of importance of biodiversity for its productive use.
CO5	Identify the global environmental problems by different types of environmental Pollution and international summits for minimizing the problems.
CO6	Outline the features of laws and rules related to environment protection, environmental impact assessment towards sustainable development.

IV. SYLLABUS:

MODULE–I: ECOSYSTEMS

Environment: definition, scope and importance of ecosystem, classification, structure and function of an ecosystem, food chains, food webs and ecological pyramids, flow of energy; biogeochemical cycles, hydrological cycle, phosphorous cycle, nitrogen cycle, biomagnifications.

MODULE-II: NATURAL RESOURCES

Natural resources: classification of resources, living and nonliving resources; water resources: use and over utilization of surface and ground water, floods and droughts, dams, benefits and problems; mineral resources: use and exploitation, environmental effects of extracting and using mineral resources; land resources; energy resources: renewable and non-renewable energy sources, use of alternate energy source.

MODULE-III: BIODIVERSITY AND BIOTIC RESOURCES

Biodiversity and biotic resources: introduction, definition, genetic, species and ecosystem diversity; value of biodiversity: consumptive use, productive use, social, ethical, aesthetic and optional values; Hot spots of biodiversity.
Threats to biodiversity: habitat loss, poaching of wildlife, human-wildlife conflicts; Conservation of biodiversity: In situ and ex situ conservation.

MODULE–IV: ENVIRONMENTAL POLLUTION AND CONTROL TECHNOLOGIES
Environmental pollution: definition, causes, effects and control measures of air pollution, water pollution, soil pollution, impacts of modern agriculture and noise pollution; solid waste: municipal solid waste management, composition and characteristics of e-waste and its management; Pollution control technologies: waste water treatment methods, primary, secondary and tertiary; global environmental issues and global efforts: climate change and impacts on human environment, ozone depletion, ozone depleting substances; International conventions / protocols: Kyoto protocol and Montreal protocol.

MODULE–V:ENVIRONMENTAL POLICY AND LEGISLATION
Environmental legislations: environmental protection act, air act1981, water act, forest act. municipal solid waste management and handling rules, biomedical waste management and handling rules, hazardous waste management and handling rules, population and its explosion.

V. TEXT BOOKS:

1. Erach Bharucha, *Text Book of Environmental Studies for Under Graduate Course*, Orient Black Swan, 3rd Edition, 2021.
2. Anubha Kaushik and C P Kaushik, *Perspectives in Environmental Studies*, New Age International private limited, New Delhi,7th Edition, 2021.
3. Benny Joseph, *Environmental Studies*, Tata Mc Graw Hill Publishing Co. Ltd, New Delhi, 3rd Edition, 2017.

VI. REFERENCE BOOKS:

1. Dr.M Anji Reddy, *Text Book of Environmental Science and Technology*, BS Publications, 3rd Edition, 2014.
2. Y Anjaneyulu, *Introduction to Environmental Science*, BSP Books Private Limited, 3rd Edition, 2020.

VII.ELECTRONICSRESOURCES:

1. https://www.meripustak.com/Environmental-Science-Isv-8th-Edition-121505
2. https://www.meripustak.com&amp;gclid=CjwKCAjwtp2bBhAGEiwAOZZTuFwLEkGc6SGNUZjXpz0ffeNwgBOHWQIKge-E-9UvXxTPxQJdjaTgJBoCrQIQAvD_BwE

VIII. MATERIALS ONLINE

1. Course Template
2. Tutorial Question Bank
3. Model Question Paper – I
4. Model Question Paper - II
5. Lecture Notes
6. Early Lecture Readiness Videos
7. Power Point Presentation

COURSE CONTENT

ENGINEERING CHEMISTRY								
I Semester: CSE / CSE (CS) / CSE (DS) **II Semester:** AE / ME / CE / ECE / EEE / CSE (AI & ML) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSD03	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	-	-	3	40	60	100
Contact Classes: 64	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 64**		
Prerequisite: Basic principles of chemistry								

I. COURSE OVERVIEW:

This course focuses on the fundamental concepts of chemistry and then builds an interface with their industrial applications. The basic knowledge on chemical bonding and intermolecular forces which together are responsible for determining the properties of materials. The students will be able to analyze water purification processes to avoid industrial interruptions. The course concludes with an overview of involving electron transfer, including their applications in corrosion and energy storage for portable electronic devices. It should cultivate in students to identify chemistry in each piece of finely engineered products used in households and industry.

II. COURSES OBJECTIVES:

The students will try to learn

I. The concepts of electrochemical principles and causes of corrosion in the new developments and breakthroughs efficiently in engineering and technology.
II. The different parameters to remove causes of hardness of water and their reactions towards complexometric method.
III. The properties, separation techniques of natural gas and crude oil along with potential applications in major chemical reactions.
IV. The different types of materials with respect to mechanisms and its significance in industrial applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Acquire the basic knowledge of electrochemical principles related to corrosion and its control
CO2	Interpret the basic properties of water for its usage in industrial and domestic applications.
CO3	Use complexometry for calculation of hardness of water to avoid industrial problems.
CO4	Extend the applications of polymers based on their degradability and properties.
CO5	Choose the appropriate fuel based on their calorific value for energy efficient processes.
CO6	Predict the knowledge on viability of advanced materials for technological improvements in various sectors.

IV. COURSE CONTENT:

MODULE-I: BATTERIES CHEMISTRY AND CORROSION (10)

Introduction to electrochemical cells: Galvanic cell, electrolytic cell; electrochemical series and its applications; Batteries: classification of batteries, construction, working and applications of Zinc-air battery, Lead-acid battery, Li-ion battery, applications of Li-ion battery to electric vehicles; Corrosion: causes and effects of corrosion, theories of chemical and electrochemical corrosion, mechanism of electrochemical corrosion; Corrosion control methods: cathodic protection, sacrificial anode and impressed current methods; Metallic coatings: Galvanization and tinning, electroplating of Copper.

MODULE-II: WATER AND ITS TREATMENT (09)
Introduction: Hardness of water, causes of hardness; types of hardness, temporary and permanent hardness, expression and units of hardness; estimation of hardness of water by complexometric method; potable water and its specifications, steps involved in the treatment of water, disinfection of water by chlorination and ozonization; external treatment of water; ion-exchange process; desalination of water: reverse osmosis, numerical problems.

MODULE-III: POLYMER TECHNOLOGY (10)
Polymers: classification of polymers; types of polymerization-addition, condensation polymerization with examples. Plastics: thermoplastic and thermosetting plastics; preparation, properties and engineering applications of PVC, Nylon 6,6 and Bakelite.
Biodegradable polymers: polylactic acid and polyvinyl alcohol and their applications. Elastomers: Introduction to natural rubber, vulcanization of natural rubber, preparation, properties and engineering applications of Buna-S and Thiokol rubber.

MODULE–IV: ENERGY SOURCES (09)
Introduction to fuels; classification of fuels; Solid fuels: coal; analysis of coal, proximate and ultimate analysis and their significance; Liquid fuels: petroleum and its refining; Gaseous fuels: composition, characteristics and applications of natural gas, LPG and CNG; Alternative and non-conventional sources of energy: solar, wind and hydropower advantages and disadvantages. Calorific value of fuel: HCV and LCV, Dulongs formula, calculation of air quantity required for complete combustion of fuel, numerical problems.

MODULE–V: ENGINEERING MATERIALS (10)
Nanomaterials: Introduction, preparation of nanoparticles by sol-gel method, chemical reduction method
and applications of nanomaterials. Smart materials and their engineering applications: shape memory
materials, Poly L-Lactic acid. Thermoresponse materials: Polyacryl amides, Poly vinyl amides.
Cement: composition of Portland cement, setting and hardening of cement.

Lubricants: characteristics of a good lubricant, mechanism of lubrication, thick film, thin film and extreme
pressure lubrication; properties of lubricants: viscosity, flash and fire point, cloud and pour point.

V. TEXT BOOKS:
1. JAIN &JAIN, P.C. Jain, Monika Jain, *Engineering Chemistry* , Dhanpat Rai publishing Company (P) limited, 17th edition, 2022.

VI. REFERENCE BOOKS:
1. Shashi Chawla, *Text Book of Engineering Chemistry*, Dhanat Rai and Company (P) Limited, 1st Edition, 2017.
2. Jayashree Anireddy, *Textbook of Engineering chemistry*, Wiley Publications, 2023
3. S.S.Dara, *Text of Engineering Chemistry*, S.Chand & Co, New Delhi, 12th edition, 2018.
4. Nitin K Puri**,** *Nanomaterials Synthesis Properties and Applications***,** I K international publishing house pvt Ltd,**1st edition** 2021**.**

VII. ELECTRONICS RESOURCES:
1. Engineering chemistry (NPTEL Web-book), by B. L. Tembe, Kamaluddin and M. S.Krishnan.http://www.cdeep.iitb.ac.in/webpage_data/nptel/Core%20Science/Engineering%20Chemistry%201/About- Faculty.html
2. https://books.google.co.in/books?id=R1JtyILNIsAC&pg=PR3&source=gbs_selected_pages&cad=3#v=onepage&q&f=false
3. https://books.google.co.in/books?id=eQTLCgAAQBAJ&pg=SA1PA53&source=gbs_selected_pages&cad=3#v=onepage&q&f=false

VIII. MATERIALS ONLINE

1. Course Template
2. Tutorial Question Bank
3. Tech talk Topics
4. Assignments
5. Definition and Terminology
6. Model Question Paper – I
7. Model Question Paper - II
8. Lecture Notes
9. Early Lecture Readiness Videos
10. Power point presentation

COURSE CONTENT

APPLIED PHYSICS									
I Semester: CSE / CSE (CS) / CSE (DS) II Semester: AE / ME / CE / ECE / EEE / CSE (AI & ML) / IT									
Course Code	Category	Hours / Week			Credits	Maximum Marks			
AHSD07	Foundation	L	T	P	C	CIA	SEE	Total	
		3	-	-	3	40	60	100	
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48			
Prerequisite: Basic principles of physics									

I. COURSE OVERVIEW:

The aim of this course is to enhance understanding of fundamental knowledge in physics needed for the future technological advances. The framework prepares students to engage in scientific questioning and extend thinking to investigations. The concepts cover current topics in the fields of solid state physics, modern physics, superconductors and nanotechnology. This knowledge helps to develop the ability to apply the principles in many technological sectors such as nanotechnology, optical fiber communication, quantum technology etc.

II. COURSES OBJECTIVES:

The students will try to learn

V. Fundamental concepts needed to explain a crystal structure in terms of atom positions, unit cells, and crystal symmetry.

VI. Basic formulations in wave mechanics for the evolution of energy levels and quantization of energies for a particle in a potential box with the help of mathematical description.

VII. The metrics of optoelectronic components, lasers, optical fiber communication and be able to incorporate them into systems for optimal performance.

VIII. The appropriate magnetic, superconducting and nanomaterials required for various engineering applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Use the general rules of indexing of directions and planes in lattices to identify the crystal systems and the Bravais lattices.
CO 2	Use the concepts of dual nature of matter and Schrodinger wave equation to a particle enclosed in simple systems.
CO 3	Analyze the concepts of laser with normal light in terms of mechanism for applications in different fields and scientific practices.
CO 4	Strengthen the knowledge on functionality of components in optical fiber communication system by using the basics of signal propagation, attenuation and dispersion.
CO 5	Gain deeper understanding on properties of magnetic and superconducting materials suitable for engineering applications.
CO 6	Review the principle factors, fabrication, characterization techniques and the applications of nanomaterials.

IV. COURSE CONTENT:

MODULE - I: CRYSTAL STRUCTURES (10)

Introduction, space lattice, basis, unit cell, lattice parameter, Bravais lattices, crystal systems, structure and packing fractions of simple cubic, body centered cubic, face centered cubic crystals, directions and planes in crystals, Miller indices, separation between successive [h k l] planes.

MODULE –II: QUANTUM PHYSICS (09)

Waves and particles, de Broglie hypothesis, matter waves, Davisson and Germer’s experiment, Schrödinger’s time independent wave equation, physical significance of the wave function, infinite square well potential.

MODULE –III: LASERS AND FIBER OPTICS (10)
Characteristics of lasers, spontaneous and stimulated emission of radiation, population inversion, lasing action, Ruby laser, He-Ne laser, applications of lasers.

Principle and construction of an optical fiber, acceptance angle, numerical aperture, types of optical fibers (Single mode, multimode, step index, graded index), optical fiber communication system with block diagram, applications of optical fibers.

MODULE –IV: MAGNETIC AND SUPERCONDUCTING PROPERTIES (10)
Permeability, field intensity, magnetic field induction, magnetization, magnetic susceptibility, origin of magnetic moment, Bohr magneton, classification of dia, para and ferro magnetic materials on the basis of magnetic moment, Hysteresis curve.
Superconductivity, general properties, Meissner effect, effect of magnetic field, type-I & type-II superconductors, BCS theory, applications of superconductors.

MODULE –V: NANOTECHNOLOGY (09)
Nanoscale, quantum confinement, surface to volume ratio, bottom-up fabrication: Sol-gel, precipitation, combustion methods, top-down fabrication: Ball milling, physical vapor deposition, chemical vapor deposition, characterization techniques: X-ray diffraction, transmission electron microscopy, applications of nanomaterials.

V. TEXT BOOKS:

2. Arthur Beiser, Shobhit Mahajan and Rai Choudhary, *Concepts of Modern Physics*, Tata McGraw Hill, 7th Edition, 2017.

VI. REFERENCE BOOKS:

2. H J Callister, *A Textbook of Materials Science and Engineering*, Wiley Eastern Edition, 8th Edition, 2013.
3. Halliday, Resnick and Walker, *Fundamentals of Physics*, John Wiley &Sons,11th Edition, 2018.
4. Charles Kittel, *Introduction to Solid State Physics,* Wiley Eastern, 2019.
5. S.L. Gupta and V. Kumar, *Elementary Solid State Physics,* Pragathi Prakashan, 2019.
6. K K Chattopadhyay and A N Banerjee, *Introduction to Nanoscience and Nanotechnology*, Prentice Hall India, 2nd Edition, 2011.

VII. ELECTRONICS RESOURCES:

4. NPTEL :: Physics - NOC:Quantum Mechanics I
5. NPTEL :: Physics - NOC:Introduction to Solid State Physics
6. NPTEL :: Physics - NOC:Solid State Physics
7. https://nptel.ac.in/courses/104104085
8. NPTEL :: Metallurgy and Material Science - NOC: Nanotechnology, Science and Applications

VIII. MATERIALS ONLINE

11. Course template
12. Tutorial question bank
13. Definition and terminology
14. Tech-talk topics
15. Assignments
16. Model question paper - I
17. Model question paper - II
18. Lecture notes
19. Early learning readiness videos (ELRV)
20. Power point presentations

COURSE CONTENT

DIFFERENTIAL EQUATIONS AND VECTOR CALCULUS								
II Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours/Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AHSD08	**Foundation**	3	1	-	4	40	60	100
Contact Classes: 48	**Tutorial Classes: 16**	**Practical Classes: Nil**				**Total Classes: 64**		
Prerequisite: Basic Principles of Matrices and Calculus								

I. COURSE OVERVIEW:

This course serves as a foundation course on differential equations and vector calculus. It includes techniques for solving ordinary differential equations, partial differential equations, vector differentiation and vector integration. It is designed to extract the mathematical developments, skills, from basic concepts to advance level of engineering problems to meet the technological challenges.

II. COURSE OBJECTIVES:

The students will try to learn:

I	The analytical methods for solving first and higher order differential equations with constant coefficients.
II	The analytical methods for formation and solving partial differential equations.
III	The physical quantities of vector valued functions involved in engineering field.
IV	The logic of vector theorems for finding line, surface and volume integrals.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Utilize the methods of differential equations for solving the orthogonal trajectories and Newton's law of cooling. cooling.
CO2	Solve the higher order linear differential equations with constant coefficients by using method of variation of parameters.
CO3	Make use of analytical methods for PDE formation to solve boundary value problems.
CO4	Identify various techniques of Lagrange's method for solving linear partial differential equations which occur in science and engineering.
CO5	Interpret the vector differential operators and their relationships for solving engineering problems.
CO6	Apply the integral transformations to surface, volume and line of different geometrical models in the domain of engineering.

IV. COURSE CONTENT:

MODULE-I: FIRSTORDERAND FIRST DEGREE ODE (10)

Exact differential equations, Equations reducible to exact differential equations, linear and Bernoulli's equations, Applications: Orthogonal Trajectories (Cartesian Coordinates) Newton's law of cooling.

MODULE-II: ORDINARY DIFFERENTIAL EQUATIONS OF HIGHER ORDER (10)

Second order linear differential equations with constant coefficients: non-homogeneous terms of the type e^{ax}, sinax, cosax, polynomials in x, $e^{ax}(x)$and method of variation of parameters.

MODULE-III: PARTIAL DIFFERNTIATIAL EQUATIONS (09)

Formation of partial differential equations by elimination of arbitrary constants and arbitrary functions.

Solutions of first order linear equations: method of grouping and method of multipliers.

MODULE–IV: VECTOR DIFFERNTIATION (09)

Scalar and vector point functions; definitions of gradient, divergent and curl with examples; solenoidal and irrotational vector point functions; scalar potential function.

MODULE–V: VECTOR INTEGRATION (10)

Line integral, surface integral and volume integral, Green‘s theorem in a plane, Stoke’s theorem and Gauss divergence theorem without proofs.

V. TEXT BOOKS:

1. B. S. Grewal, *Higher Engineering Mathematics*, 44/e, Khanna Publishers, 2017.
2. Erwin Kreyszig, *Advanced Engineering Mathematics*, 10/e, John Wiley & Sons, 2011.

VI. REFERENCE BOOKS:

1. R. K. Jain and S. R. K. Iyengar, *Advanced Engineering Mathematics*, 3/ed, Narosa Publications, 5th Edition, 2016.
2. George B. Thomas, Maurice D. Weir and Joel Hass, Thomas, *Calculus*, 13/e, Pearson Publishers, 2013.
3. N.P.Bali and Manish Goyal, *A text book of Engineering Mathematics*, Laxmi Publications, Reprint, 2008
4. Dean G. Duffy, *Advanced Engineering Mathematics with MATLAB*, CRC Press.
5. Peter O’Neil *Advanced Engineering Mathematics*, Cengage Learning.
6. B.V. Ramana, *Higher Engineering Mathematics*, McGraw Hill Education.

VII. ELECTRONIC RESOURCES:

1. Engineering Mathematics - I, By Prof. Jitendra Kumar |https://onlinecourses.nptel.ac.in/noc23_ma88/preview
2. Advanced Calculus for Engineers, By Prof. Jitendra Kumar, Prof. Somesh Kumar https://onlinecourses.nptel.ac.in/noc23_ma86/preview
3. http://www.efunda.com/math/math_home/math.cfm
4. http://www.ocw.mit.edu/resourcs/#Mathematics
5. http://www.sosmath.com
6. http://www.mathworld.wolfram.com

VIII. MATERIAL ONLINE:

1. Course template
2. Tech-talk topics
3. Assignments
4. Definition and terminology
5. Tutorial question bank
6. Model question paper – I
7. Model question paper – II
8. Lecture notes
9. Early lecture readiness videos (ELRV)
10. Power point presentations

COURSE CONTENT

ESSENTIALS OF PROBLEM SOLVING									
II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT / ECE / EEE									
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**			
ACSD05	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**	
		3	0	0	3	40	60	100	
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**			
Prerequisite: There is no prerequisite to take this course									

I. COURSE OVERVIEW:

This course aims to provide exposure to problem solving through programming. Useful graph theory concepts, numerical techniques, and their applications to real world problems are discussed. Graph theoretical notions and the use of algorithms, both in the mathematical theory of graphs and its applications are discussed. Student will also learn how to implement and interpret numerical solutions by writing a well-designed computer programs in regard to their efficiency and suitability for real-life applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts of graph theory and its properties.
II. The basics related to paths and cycles using Eulerian and Hamiltonian cycles.
III. The applications of graph colouring and traversal algorithms for solving real-time problems.
IV. The numerical methods to solve algebraic equations.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Outline the graph terminologies, graph representation, and relate them to practical examples.
CO2	Build efficient graph routing algorithms for various optimization problems on graphs.
CO3	Use effective techniques from graph theory to solve problems in networking and telecommunication.
CO4	Interpret the fundamental concepts of polynomials, roots of equations and solve corresponding problems using computer programs.
CO5	Apply the knowledge of numerical methods to solve algebraic and transcendental equations arising in real-life situations.
CO6	Solve numerical integrals and ordinary differential equations to simulate discrete time algorithms.

IV. COURSE CONTENT:

MODULE - I GRAPH THEORY (08)

Graph terminology, digraphs, weighted graphs, complete graphs, graph complements, bipartite graphs, graph combinations, isomorphisms, matrix representations of graphs, incidence and adjacency matrices, degree sequence.

MODULE - II GRAPH ROUTES (10)

Eulerian circuit: Konigsberg bridge problem, touring a graph; Eulerian graphs, Hamiltonian cycles, the traveling salesman problem; Shortest paths: Dijkstra's algorithm, walks using matrices.

MODULE - III GRAPH COLORING AND GRAPH ALGORITHMS (10)

Four color theorem, vertex coloring, edge coloring, coloring variations, first-fit coloring algorithm.

Graph traversal: depth-first search, bread-first search and its applications; Minimum spanning trees: Kruskal's and Prim's algorithm, union-find structure.

MODULE - IV: ALGEBRAIC AND TRANSCENDENTAL EQUATIONS (10)

Algebraic equations, method of false position, bisection method, iteration method, Newton-Raphson method, Secant method, Ramanujan’s Method, Muller’s method (Approximation up to 2 decimals only).

MODULE - V: NUMERICAL INTEGRATION AND ORDINARY DIFFERENTIATIAL EQUATIONS (10)

Trapezoidal rule, Simpson’s 1/3 rule, Simpson’s 3/8 rule, Solution by Taylor’s series, Euler’s method of solving an ordinary differential equation numerically, Runge-Kutta’s second order method of solving ordinary differential equations (Approximation up to 2 decimals only).

V. TEXT BOOKS:

2. Karin R Saoub, *Graph Theory: An Introduction to Proofs, Algorithms, and Applications*, 1st edition, Chapman and Hall, 2021.
3. S S Sastry, *Introductory Methods of Numerical Analysis*, 5th edition, 2012.

VI. REFERENCE BOOKS:

1. Mahinder Kumar Jain, *Numerical Methods: For Scientific and Scientific Computation*, New Age International Pvt. Ltd., 7th edition, 2019.
2. P Kandasamy, K Thilagavathy, K Gunavathi, *Numerical Methods*, S Chand and Company, 2006.
3. R Balakrishnan, K Ranganathan, *A Textbook of Graph Theory*, Springer Exclusive, 2nd edition, 2019.
4. Jann Kiusalaas, *Numerical Methods in Engineering with Python*, Cambridge University Press, 2nd edition 2010.
5. Gary Chartrand, Ping Zhang, *A First Course in Graph Theory*, Dover Publications Inc., 2012.
6. James F. Epperson, *An Introduction to Numerical Methods and Analysis*, Wiley, 2nd edition, 2013.

VII. ELECTRONICS RESOURCES:

1. https://www.geeksforgeeks.org/numerical-methods-and-calculus-gq/
2. https://www.geeksforgeeks.org/program-for-bisection-method/
3. https://ocw.mit.edu/courses/2-993j-introduction-to-numerical-analysis-for-engineering-13-002j-spring-2005/pages/lecture-notes/
4. https://www.tutorialspoint.com/graphs-and-its-traversal-algorithms
5. https://web.mit.edu/urban_or_book/www/book/chapter6/6.4.4.html
6. https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/
7. https://www.codingninjas.com/studio/library/euler-and-hamilton-paths

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

ENGINEERING CHEMISTRY LABORATORY									
I Semester: CSE / CSE (CS) / CSE (DS) **II Semester:** AE / ME / CE / ECE / EEE / CSE (AI&ML) / IT									
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSD05	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	3	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 36**				**Total Classes: 36**		
Prerequisite: Basic Principles of Chemistry								

I. COURSE OVERVIEW:

The course encourages introducing analytical tools in an Engineering perspective. The course efforts to provide the basic knowledge of analytical methodology, outlines the importance of volumetric analysis, comprehensive instrumental analysis for properties of fuels, colorimetric analysis and spectroscopic analysis. This practical approach gives the essence of analytical chemistry for skill development in determinations of materials properties and its viability in the industry.

II. COURSES OBJECTIVES:

The students will try to learn:

I. The quantitative analysis to know the strength of unknown solutions by instrumental methods.
I. The troubles of hard water and its estimation by analytical techniques.
II. The applications of appropriate lubricant for finely tuned machinery.
III. The basic knowledge on quantity of light absorbed by the materials.

III. COURSE OUTCOMES:

After successful completion of the course students should be able to:

CO1	Use conductivity meter and potentiometer for measurement of conductance and electromotive force of solutions
CO2	Examine the corrosion tendency in metals and its control by using inhibitor.
CO3	Make use of the principles of water analysis for domestic and industrial applications.
CO4	Demonstrate the characteristics of different fuels for finding the calorific value.
CO5	Use different types of lubricants to know its properties for the proper lubrication of machinery in industries.
CO6	Interpret the absorption tendency of solids or liquids by using colorimetry and spectroscopy techniques.

IV. COURSE CONTENT:

1. GETTING STARTED EXERCISES

1.1 Introduction to Chemistry Laboratory

The fundamental concepts and theories required for carrying out qualitative and quantitative analysis. Detailed explanation on the analytical techniques used for qualitative analysis. Emphasis on instrumental method of analysis and its advantages over conventional methods.

i. Types of analysis
ii. Difference between qualitative and quantitative analysis
iii. Common techniques of qualitative and quantitative analysis
iv. Introduction to instrumental method of analysis
v. Introduction to basic techniques and handling of common apparatus
vi. Discussion of Material Safety Data Sheet (MSDS) of chemicals
vii. Identification of toxic signs and safety procedures of chemical laboratory

1.2 Safety Guidelines to Chemistry Laboratory

The chemistry laboratory must be a safe place in which to work and learn about chemistry.

i. Wear a chemical-resistant apron.
ii. Be familiar with your lab work sheet before you come to lab. Follow all written and verbal instructions carefully. Observe the safety alerts in the laboratory directions. If you do not understand a direction or part of a procedure, ask the teacher before proceeding.
iii. When entering the laboratory room, do not touch any equipment, chemicals, or other materials without being instructed to do so. Perform only those experiments authorized by the instructor.
iv. If you take more of a chemical substance from a container than you need, you should not return the excess to the container. This might cause contamination of the substance remaining. Dispose of the excess as your instructor directs.
v. Never smell anything in the laboratory unless your teacher tells you it is safe. Do not smell a substance by putting your nose directly over the container and inhaling. Instead, waft the vapors toward your nose by gently fanning the vapors toward yourself.
vi. Do not directly touch any chemical with your hands. Never taste materials in the laboratory.
vii. Work areas should be kept clean and tidy at all times. Always replace lids or caps on bottles and jars.

1.3 Data recording and reports

Students must record their experimental values in the provided tables in this laboratory manual and reproduce them in the laboratory worksheets. Worksheets are integral to recording the methodology and results of an experiment. In engineering practice, the laboratory worksheets serve as a valuable reference to the technique used in the laboratory. Note that the data collected will be an accurate and permanent record of the data obtained during the experiment and the analysis of the results.

2. CONDUCTOMETRY

2.1 Determine the neutralization point between strong acid against strong base

i. **The basic principle of conductometric titrations**

ii. **Titration of unknown solution of acid with base**

iii. **Graphical plots on volume of titrant vs. conductance**

3. POTENTIOMETRY

3.1 Estimate the amount of Iron by using potentiometry

i. **The basic principle of potentiometric titrations**

ii. **Titration of Mohr's salt with potassium dichromate**

iii. Graphical plots on volume of titrant vs. potential

4. pH METRY

4.1 Determine the pH of the unknown solution by pH metry

i. **The basic principle of pH metry**

ii. **Titration of unknown solution with standard acid**

iii. Graphical plots on volume of titrant vs. pH to obtain equivalence point

5. ARGENTOMETRIC TITRATIONS

5.1 Determination of chloride content of water by argentometry

i. Principle of Argentometric titration.

ii. Titration of water samples by using EDTA to find the total hardness in water.

6. MEASUREMENT OF TOTAL DISSOLVED SOLIDS IN WATER

6.1 Measurement of total dissolved solids (TDS) in different water samples

i. **Specifications of potable water**

ii. Measure the total dissolved solids in different water samples by TDS meter

7. COMPLEXOMETRY METHOD

7.1 Estimate the total hardness of water by EDTA

i. **Principle of complexometric titration**

ii. Titration of water samples by using EDTA to find the total hardness in water.

8. BOMB CALORIMETER

8.1 Determine the calorific value of solid or liquid fuels using Bomb calorimeter

i. **Significance of bomb calorimeter**

ii. Combustion of solid or liquid fuels in bomb calorimeter to find its calorific value

9. VISCOSITY OF LUBRICANT

9.1 Determine the viscosity of the lubricants using Red Wood viscometer / Ostwald's viscometer

i. The principle of viscosity of lubricant

ii. Significance of viscosity index of lubricant

iii. Viscosity of given lubricant at various temperature by using Red wood viscometer

10. FLASH AND FIRE POINTS OF LUBRICANT

10.1 Determine the flash and fire points of lubricants

i. **Significance of flash and fire point of lubricant in industries**

ii. Flash and Fire points of a given lubricant by using Pensky Martens flash point apparatus

11. CLOUD AND POUR POINTS OF LUBRICANT

11.1 Determine cloud and pour points of lubricants

i. Significance of cloud and pour point of lubricants in industries

ii. Cloud and Pour points of a given lubricant by using cloud and pour point apparatus

12. COLORIMETRY

12.1 Estimate the metal ion concentration using colorimeter

i. Complexation of metal ion with ligands

ii. Detection of absorbance of the colored metal -ligand complex solution

iii. Graphical determination of concentration of the metal ions in the solution

13. SPECTROSCOPY

13.1Characterization of nanomaterials by UV-visible spectrophotometer

i. Synthesis of silver oxide nanomaterials

ii. Dispersion of nanoparticles in suitable solvent

iii. Determination of absorption edge of the nanoparticles using spectroscopic technique

V. TEXTBOOKS:

1. K. Mukkanti et al. *Practical Engineering Chemistry*, B.S. Publications, Hyderabad.
2. Vogel's, *Quantitative chemical analysis*, prentice Hall, 6th Edition, 2009.

VI. REFERENCE BOOKS:

1. Solanki, M. K. Engineering Chemistry Laboratory Manual. (Edu creation Publishing, 2019).
2. Jeffery, G. H. in TEXTBOOK OF QUANTITATIVE CHEMICAL ANALYSIS (ed John Wiley and Sons) (1989).
3. Gary-D-Christian, P. K. S. D., Kevin A. Schug. Analytical-Chemistry-by-Gary-D-Christian. 7 edn, Vol. 7 826 (Wiley, 2014).
4. Budinski, Kenneth G., Engineering materials: properties and selection, 5th edition, Prentice-Hall, 1996, pg.423.
5. Engineering chemistry by Jain & Jain, 17th Edition, ISBN:978-93-5216-641-1
6. Nitin K Puri, "Nanomaterials synthesis properties and applications" I K international publishing house Pvt Ltd, 1st Edition 2021**.**
7. B. Ramadevi and P. Aparna, S Chand publications, *lab manual for engineering chemistry*, S Chand publications, NewDelhi,1st Edition 2022.

VII. ELECTRONICS RESOURCES:

1. https://nptel.ac.in/translation
2. https://nptel.ac.in/courses/115105120
3. https://archive.nptel.ac.in/courses/122/101/122101001/#

VIII. MATERIALS ONLINE

1. Course Template
2. Laboratory Manual

COURSE CONTENT

APPLIED PHYSICS LABORATORY								
I Semester: CSE / CSE (CS) / CSE (DS) **II Semester:** AE / ME / CE / ECE / EEE / CSE (AI & ML) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSD09	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Basic Principles of Physics								

I. COURSE OVERVIEW:

The aim of the course is to provide hands on experience for experiments in different areas of physics. Students will be able to perform the experiments with interest and an attitude of learning. This laboratory includes experiments involving electromagnetism and optoelectronics. These also develop student's expertise in applying physical concepts to practical problem and apply it for different applications.

II. COURSES OBJECTIVES:

The students will try to learn:

I	Familiarize with the lab facilities, equipment, standard operating procedures.
II	The different kinds of functional magnetic materials which paves a way for them to usein various technical and engineering applications.
III	The analytical techniques and graphical analysis to study the experimental data for optoelectronicdevices.
IV	The application of characteristics of lasers and its propagation in optical fiber communication.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO1	Identify the type of semiconductor using the principle of Hall effect and also determine the energy gap and resistivity of a semiconductor diode using four probe methods.
CO2	Illustrate principle, working and application of wave propagation and compare the results of frequency with theoretical harmonics and overtones.
CO3	Investigate the energy losses, curie temperature and properties associated with a given Ferro magnetic material.
CO4	Examine launching of light through optical fiber from the concept of light gathering capacity of numerical aperture and determine the divergence of Laser beam
CO5	Graph V-I /L-I characteristics of various optoelectronic devices like Light Emitting diode, Solar cell at different intensities to understand their basic principle of functioning as well as to infer the value of Planck's constant.
CO6	Analyze the variation of magnetic field induction produced at various points along the axis of current carrying coil.

IV. COURSE CONTENT:

1. GETTING STARTED EXCERCISES

1.1 On Errors and Uncertainty in a measurement:

When a number represents a physical measurement, it is never exact because of the limitations of the instrument used or the way it was employed etc. It is essential, therefore, that each experimental result be presented in a way that indicates its reliability. The accuracy of result is important, for example, the calibration of the measuring instruments or systematic errors on the part of whoever is taking the data.

The following table is useful in thinking about these concepts:

Problem	Remedy
Mistakes and blunders	Repeat measurements several times to check yourself
Systematic errors	Use calibrated instruments properly and carefully
Random errors	Treat data statistically and report on the average magnitude of errors

1.2 Making a good graph:

- Keep your axes straight: If you need to plot "A vs B", or "A as a function of B", then A is on the vertical axis and B is on the horizontal axis.
- The crucial part is choosing the range and scale for each axis. The range must be just large enough to accommodate all data and small enough that the scale is readable.
- The scale should be spread out enough so that data take up most of the graph area and labeled so that plotting (and reading) is easy. Where appropriate, error bars should be included to indicate the uncertainty in measurements.
- When a line is drawn, it should be a smooth one that best fits data. In general, there should be as many points on one side of the line as on the other. If data is taken properly, the line should pass inside of the error bars for each point.
- If graph shows that one quantity is proportional to another, it should be a straight line that starts at the origin and passes through the plotted data with as many points on one side as the other.
- If the slope of the line is to be found, choose two points on the line that are as far apart as possible. This will minimize the error that is introduced in reading the value of those points.
- The slope is the difference between the vertical values of those points divided by the difference in the horizontal values of those points.

1.3 Data recording and worksheets

- Write the work sheets for the allotted experiment and keep them ready before the beginning of each lab.
- Perform the experiment and record the observations in the worksheets.
- Analyze the results and get the work sheets evaluated by the faculty.
- Upload the evaluated reports online from CMS LOGIN within the stipulated time.

2. HALL EFFECT (LORENTZ FORCE)

2.1 Study the phenomenon of Hall effect and determine the charge carrier density and Hall coefficient of a given sample.

2.2 Hint whether the given semiconductor is p - type or n - type using the principle of Hall Effect.

ENERGY GAP OF A SEMICONDUCTOR DIODE

2.3 Determination of energy gap of a given semiconductor diode by measuring the variation of current as a function of temperature.

2.4 Try to find the Fermi level of the given semiconductor

3. RESISTIVITY – FOUR PROBE METHOD

3.1 Determination of the resistivity by forcing current through two outer probes

3.2 Formulate the reading of voltage across the two inner probes of semiconductor by four probe method.

4. MELDE'E EXPERIMENT

4.1 Determination of frequency of a given tuning fork in longitudinal wave propagation.

4.2 Try to establish the transverse mode of wave propagation by understanding the theoretical harmonics and overtones.

5. B-H CURVE WITH CRO

5.1 Evaluate the energy loss per unit volume of a given magnetic material per cycle by tracing the hysteresis loop (B-H curve).

5.2 Observe the hysteresis loss of ferro magnetic materials.

6. MAGNETIC MATERIAL

6.1 Determine the curie temperature (Tc) of a ferromagnetic material.

6.2 Evaluate the relative permeability (μ_r) of a ferromagnetic material.

7. OPTICAL FIBER

7.1 Determine the numerical aperture of a given optical fiber.

7.2 Calculate the acceptance angle of a given optical fiber.

8. LASER DIVERGENCE

8.1 Determination of the beam divergence of the given laser beam.

8.2 Try to estimate the laser output

SOLAR CELL

8.3 Studying the characteristics of solar cell at different intensities

8.4 Try to get the maximum workable power.

9. LIGHT EMITTING DIODE

9.1 Studying V-I characteristics of LED in forward bias for different LEDs.

9.2 Measure the threshold voltage and forward resistance, and try for the dynamic Resistance

PLANCK'S CONSTANT

9.3 Determination of Planck's constant by measuring threshold voltage of given LED.

9.4 Draw the L -I characteristics of the given LED.

10. STEWART GEE'S APPARATUS

10.1 Study the magnetic field along the axis of current carrying coil – Stewart and Gee's method.

10.2 Estimate the magnetic lines of force.

11. BIASING OF DIODE

11.1 Study the forward bias of LED

11.2 Study the reverse bias of Photo diode

V. TEXT BOOKS:

1. Laboratory Experiments in College Physics", C.H. Bernard and C.D. Epp, John Wiely and Sons, Inc., New York, 1995.

VI. REFERENCE BOOKS:

1. C. L. Arora, "Practical Physics", S. Chand & Co., New Delhi, 3rd Edition, 2012.
2. Vijay Kumar, Dr. T Radhakrishna, "Practical Physics for Engineering Students", SM Enterprises, 2nd Edition, 2014.
3. Dr. Rizwana, "Engineering Physics Manual", Spectrum Techno Press, 2018.

VII. ELECTRONICS RESOURCES:

4. https://nptel.ac.in/translation
5. https://nptel.ac.in/courses/115105120
6. NPTEL:: courses-Sem 1 and 2 - Engineering Physics and Applied Physics I
7. Experimental Physics I - Course (nptel.ac.in)
8. NPTEL:: Physics - Waves and Oscillations

VIII. MATERIALS ONLINE:

1. Course template
2. Lab manual

COURSE CONTENT

PROGRAMMING FOR PROBLEM SOLVING LABORATORY								
II Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACSD06	Foundation	L	T	P	C	CIA	SEE	Total
		0	1	2	2	40	60	100
Contact Classes: Nil	Tutorial Classes: 15	Practical Classes: 30				Total Classes: 45		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

The course is designed with the fundamental programming skills and problem-solving strategies necessary to tackle a wide range of computational challenges. Through hands-on programming exercises and projects, students will learn how to write code, analyze problems and develop solutions using various programming languages and tools. The course will cover fundamental programming concepts and gradually progress to more advanced topics.

II. COURSE OBJECTIVES

The students will try to learn:

I. The fundamental programming constructs and use of collection data types in Python.
II. The ability to develop programs using object-oriented features.
III. Basic data structures and algorithms for efficient problem-solving.
IV. Principles of graph theory and be able to apply their knowledge to a wide range of practical problems across various disciplines.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Adapt programming concepts, syntax, and data structures through hands on coding exercises.
CO2	Develop the ability to solve a variety of programming problems and algorithms using python.
CO3	Implement complex and custom data structures to solve real-world problems.
CO4	Demonstrate proficiency in implementing graph algorithms to solve variety of problems and scenarios.
CO5	Develop critical thinking skills to solve the various real-world applications using graph theory.
CO6	Learn the importance of numerical methods and apply them to tackle a wide range of computational problems.

IV. COURSE CONTENT:

EXERCISES FOR PROGRAMMING FOR PROBLEM SOLVING LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Input: nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

Explanation: Because nums[0] + nums[1] == 9, so return [0, 1].

Input: nums = [3, 2, 4], target = 6

Output: [1, 2]

Input: nums = [3, 3], target = 6

Output: [0, 1]

Hints:

```
def twoSum(self, nums: List[int], target: int) -> List[int]:
    a=[]
    # Write code here
    …

    return a
```

1.2 Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Input: nums = [1, 2, 3, 1]

Output: true

Input: nums = [1, 2, 3, 4]

Output: false

Input: nums = [1, 1, 1, 3, 3, 4, 3, 2, 4, 2]

Output: true

Hints:

```
def containsDuplicate(self, nums):
      a = set() # set can have only distinct elements
      # Write code here
      …
      return False
```

1.3 Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:
I can be placed before V (5) and X (10) to make 4 and 9.
X can be placed before L (50) and C (100) to make 40 and 90.
C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Input: s = "III"

Output: 3

Input: s = "LVIII"

Output: 58

Hints:

```
def romanToInt(self, s: str) -> int:
        # Write code here
       …

        return number
```

1.4 Plus One

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's. Increment the large integer by one and return the resulting array of digits.

Input: digits = [1, 2, 3]

Output: [1, 2, 4]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1, 2, 4].

Hints:

```
def plusOne(self, digits: List[int]) -> List[int]:
       n = len(digits)
       # Write code here
        …

       return digits
```

1.5 Majority Element

Given an array nums of size n, return the majority element. The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Input: nums = [3, 2, 3]

Output: 3

Input: nums = [2, 2, 1, 1, 1, 2, 2]

Output: 2

Hints:

```
def majorityElement(self, nums):
       # write code here
              …
```

1.6 Richest Customer Wealth

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return the wealth that the richest customer has. A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Input: accounts = [[1, 2, 3], [3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Input: accounts = [[1, 5], [7,3],[3,5]]

Output: 10

Explanation:

1st customer has wealth = 6

2nd customer has wealth = 10

3rd customer has wealth = 8

The 2nd customer is the richest with a wealth of 10.

Input: accounts = [[2,8,7],[7,1,3],[1,9,5]]

Output: 17

Hints:

```
def maximumWealth(self, accounts: List[List[int]]) -> int:
    # write code here
        …
```

1.7 Fizz Buzz

Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5.

answer[i] == "Fizz" if i is divisible by 3.

answer[i] == "Buzz" if i is divisible by 5.

answer[i] == i (as a string) if none of the above conditions are true.

Input: n = 3

Output: ["1","2","Fizz"]

Input: n = 5

Output: ["1","2","Fizz","4","Buzz"]

Input: n = 15

Output: ["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","FizzBuzz"]

Hints:

```
def fizzBuzz(self, n: int) -> List[str]:
    # write code here
        …
```

1.8 Number of Steps to Reduce a Number to Zero

Given an integer num, return the number of steps to reduce it to zero. In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Input: num = 14

Output: 6

Explanation:

- 14 is even; divide by 2 and obtain 7.
- 7 is odd; subtract 1 and obtain 6.
- 6 is even; divide by 2 and obtain 3.
- 3 is odd; subtract 1 and obtain 2.
- 2 is even; divide by 2 and obtain 1.
- 1 is odd; subtract 1 and obtain 0.

Input: num = 8

Output: 4

Explanation:

- 8 is even; divide by 2 and obtain 4.
- 4 is even; divide by 2 and obtain 2.
- 2 is even; divide by 2 and obtain 1.
- 1 is odd; subtract 1 and obtain 0.

Input: num = 123

Output: 12

Hints:

```
def numberOfSteps(self, n: int) -> int:
        # write code here
                …
```

1.9 Running Sum of 1D Array

Given an array nums. We define a running sum of an array as runningSum[i] = sum (nums[0]…nums[i]).

Return the running sum of nums.

Input: nums = [1, 2, 3, 4]

Output: [1, 3, 6, 10]

Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].

Input: nums = [1, 1, 1, 1, 1]

Output: [1, 2, 3, 4, 5]

Explanation: Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1].

Input: nums = [3, 1, 2, 10, 1]

Output: [3, 4, 6, 16, 17]

Hints:

```
def runningSum(self, nums: List[int]) -> List[int]:
                # write code here
                  …
                return answer
```

1.10 Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val. Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Input: nums = [3, 2, 2, 3], val = 3

Output: 2, nums = [2, 2, _, _]

Explanation: Your function should return k = 2, with the first two elements of nums being 2.

It does not matter what you leave beyond the returned k (hence they are underscores).

Input: nums = [0,1,2,2,3,0,4,2], val = 2

Output: 5, nums = [0,1,4,0,3,_,_,_]

Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.

Note that the five elements can be returned in any order.

It does not matter what you leave beyond the returned k (hence they are underscores).

Hints:

```
def removeElement(self, nums: List[int], val: int) -> int:
            # write code here
              …
            return len(nums)
```

2. Matrix Operations

2.1 Add Two Matrices

Given two matrices X and Y, the task is to compute the sum of two matrices and then print it in Python.

Input:

```
 X= [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]

Y = [[9, 8, 7],
      [6, 5, 4],
      [3, 2, 1]]
```

Output:

```
 Result = [[10, 10, 10],
             [10, 10, 10],
             [10, 10, 10]]
```

Hints:

```
# Program to add two matrices using nested loop

X = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]

Y = [[9,8,7],
    [6,5,4],
    [3,2,1]]


result = [[0,0,0],
         [0,0,0],
         [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # write code here
    …

for r in result:
    print(r)
```

TRY

1. Take input as X = [[10, 20, 30],[41, 52, 63], [47, 58, 69]] Y = [[19,18,17],[66,35,49], [13,21,11]] and verify the results.

2.2 Multiply Two Matrices

Given two matrices X and Y, the task is to compute the multiplication of two matrices and then print it.

Input:

```
 X= [[1, 7, 3],
      [3, 5, 6],
      [6, 8, 9]]

Y = [[1, 1, 1, 2],
      [6, 7, 3, 0],
      [4, 5, 9, 1]]
```

Output:

```
 Result = [[55, 65, 49, 5],
           [57, 68, 72, 12],
           [90, 107, 111, 21]]
```

Hints:

```
# Program to multiply two matrices using list comprehension

# take a 3x3 matrix
A = [[12, 7, 3],
    [4, 5, 6],
    [7, 8, 9]]

# take a 3x4 matrix
B = [[5, 8, 1, 2],
    [6, 7, 3, 0],
    [4, 5, 9, 1]]

# result will be 3x4
# write code here
    …
for r in result:
    print(r)
```

TRY

1. Take input as X = [[11, 0, 30],[-41, -2, 63], [41, -5, -9]] Y = [[19,-48,17],[-6,35,19], [13,1,-9]] and verify the results.

2.3 Transpose of a Matrix

A matrix can be implemented using a nested list. Each element is treated as a row of the matrix. Find the transpose of a matrix in multiple ways.

Input: [[1, 2], [3, 4], [5, 6]]
Output: [[1, 3, 5], [2, 4, 6]]
Explanation: Suppose we are given a matrix

```
                    [[1, 2],
                     [3, 4],
                     [5, 6]]
```

Then the transpose of the given matrix will be,

```
                    [[1, 3, 5],
                     [2, 4, 6]]
```

Hints:

```
# Program to multiply two matrices using list comprehension
# take a 3x3 matrix
A = [[12, 7, 3],
    [4, 5, 6],
    [7, 8, 9]]

# result will be 3x4
# write code here
    …
for r in result:
    print(r)
```

TRY

1. Take input as X = [[11, 0, 30],[-41, -2, 63], [41, -5, -9]] and verify the results.

2.4 Matrix Product

Matrix product problem we can solve using list comprehension as a potential shorthand to the conventional loops. Iterate and find the product of the nested list and at the end return the cumulative product using function.

Input: The original list: [[1, 4, 5], [7, 3], [4], [46, 7, 3]]
Output: The total element product in lists is: 1622880

Hints:

```
# Matrix Product using list comprehension + loop

def prod(val):
   # write code here
    …

# initializing list
test_list = [[1, 4, 5], [7, 3], [4], [46, 7, 3]]
```

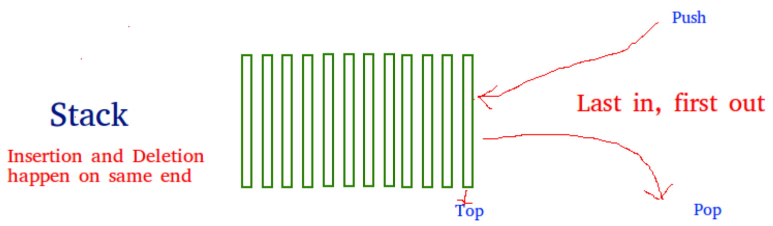
TRY

1. Take input list: [[1, 4, 5], [7, 3], [4], [46, 7, 3]] and verify the result.

3. Stack

3.1 Stack implementation using List

A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty
- **size()** – Returns the size of the stack
- **top() / peek()** – Returns a reference to the topmost element of the stack
- **push(a)** – Inserts the element ‘a’ at the top of the stack
- **pop()** – Deletes the topmost element of the stack

Hints:

```
# Stack implementation using list
```

```
top=0
mymax=5
def createStack():
    stack=[]
    return stack
def isEmpty(stack):
    # write code here
    …
def Push(stack,item):
    # write code here
    …
def Pop(stack):
    # write code here
    …
# create a stack object
stack = createStack()
while True:
    print("1.Push")
    print("2.Pop")
    print("3.Display")
    print("4.Quit")
    # write code here
    …
```

TRY

1. Take input operations as [PUSH(A),PUSH(B),PUSH(C),POP,POP,POP,POP,PUSH(D)] and verify the result.
2. Take input operations as [POP, POP, PUSH (A), PUSH (B), POP, and PUSH(C)] and verify the result.

3.2 Balanced Parenthesis Checking

Given an expression string, write a python program to find whether a given string has balanced parentheses or not.

Input: {[]{()}}

Output: Balanced

Input: [{}{}(]

Output: Unbalanced

Using stack One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return Balanced otherwise, Unbalanced.

Hints:

```
# Check for balanced parentheses in an expression
open_list = ["[","{","("]
close_list = ["]","}",")"]

# Function to check parentheses
def check(myStr):
    # write code here
    …
```

TRY

1. Take input as {[]{()}[][]} and verify the result.
2. Take input as {[]{()}[]{}} and verify the result.

3.3 Evaluation of Postfix Expression

Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.

Input: str = "2 3 1 * + 9 -"

Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = "100 200 + 2 / 5 * 7 +"

Output: 757

Procedure for evaluation postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
  - If the element is a number, push it into the stack.
  - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

Hints:

```
# Evaluate value of a postfix expression

# Class to convert the expression
class Evaluate:

    # Constructor to initialize the class variables
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity

        # This array is used a stack
        self.array = []

    # Check if the stack is empty
    def isEmpty(self):
        # write code here
        …
    def peek(self):
        # write code here
        …
    def pop(self):
        # write code here
        …
    def push(self, op):
        # write code here
        …
    def evaluatePostfix(self, exp):
        # write code here
        …
```

```
# Driver code
exp = "231*+9-"
obj = Evaluate(len(exp))

# Function call
print("postfix evaluation: %d" % (obj.evaluatePostfix(exp)))
```

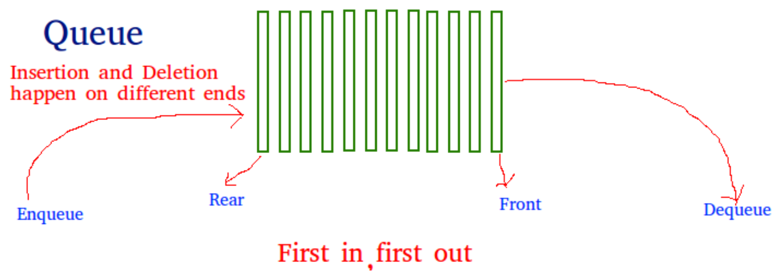
TRY

1. Take input str = “A B + C* D / E +” and verify the result.
2. Take input str = “XYZ- + W+ R / S -” and verify the result.

4. Queue

4.1 Linear Queue using List

Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



Hints:

```
# Static implementation of linear queue
front=0
rear=0
mymax=5
def createQueue():
    queue=[]   #empty list
    return queue

def isEmpty(queue):
    # write code here
    …
def enqueue(queue,item):  # insert an element into the queue
    # write code here
    …
def dequeue(queue):  #remove an element from the queue
    # write code here
    …
# Driver code
queue = createQueue()
while True:
    print("1.Enqueue")
    print("2.Dequeue")
    print("3.Display")
```

```
   print("4.Quit")
   # write code here
   …
```

TRY

1. Take input operations as [ENQUEUE(A),DEQUEUE(),ENQUEUE(B),DEQUEUE(),ENQUEUE(C),DEQUEUE(),] and verify the result.

2. Take input operations as [ENQUEUE(A), ENQUEUE(B),DEQUEUE(),ENQUEUE(C), DEQUEUE(),ENQUEUE(D), DEQUEUE(), ENQUEUE(C),DEQUEUE(),] and verify the result.

4.2 Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Input:

["MyStack", "push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 2, 2, false]

Hints:

```
class MyStack:

   def __init__(self):
     # write code here
     …
   def push(self, x: int) -> None:
     # write code here
     …
   def pop(self) -> int:
     # write code here
     …
   def top(self) -> int:
     # write code here
     …
   def empty(self) -> bool:
     # write code here
     …
# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

4.3 Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Input:

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 1, 1, false]

Hints:

```
class MyQueue:

    def __init__(self):
        # write code here
        …
    def push(self, x: int) -> None:
        # write code here
        …
    def pop(self) -> int:
        # write code here
        …
    def peek(self) -> int:
        # write code here
        …

    def empty(self) -> bool:
        # write code here
        …
# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

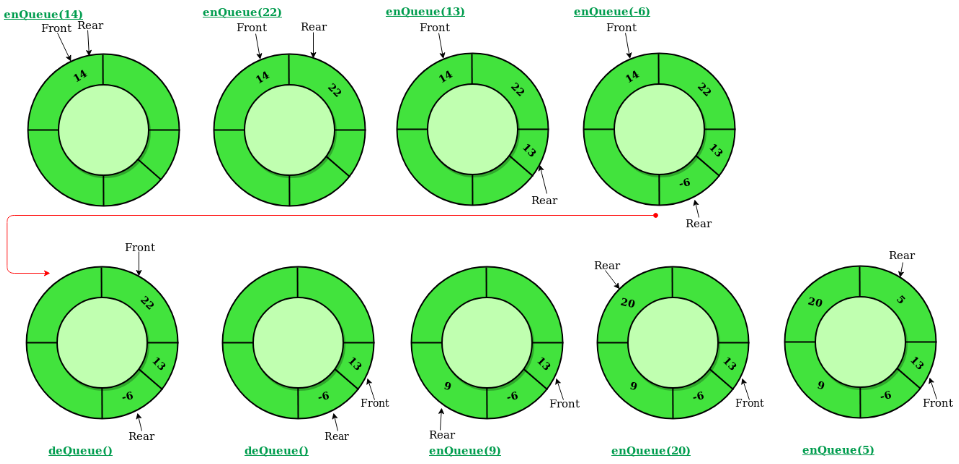
4.4 Circular Queue

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.

- Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
- If it is full then display Queue is full.
- If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
- Check whether the queue is Empty.
- If it is empty then display Queue is empty.
- If the queue is not empty, then get the last element and remove it from the queue.



Implement Circular Queue using Array:

1. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
2. Initialize two variables front and rear to -1.
3. **Enqueue:** To enqueue an element **x** into the queue, do the following:
   - Increment rear by 1.
   - If **rear** is equal to n, set **rear** to 0.
   - If **front** is -1, set **front** to 0.
   - Set queue[rear] to x.
4. **Dequeue:** To dequeue an element from the queue, do the following:
   - Check if the queue is empty by checking if **front** is -1.
   - If it is, return an error message indicating that the queue is empty.
   - Set **x** to queue [front].
   - If **front** is equal to **rear**, set **front** and **rear** to -1.
   - Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
   - Return x.

Hints:

```
class CircularQueue():

  # constructor
  def __init__(self, size): # initializing the class
    self.size = size

    # initializing queue with none
    self.queue = [None for i in range(size)]
    self.front = self.rear = -1

  def enqueue(self, data):
     # Write code here
         …

  def dequeue(self):
     # Write code here
         …
    def display(self):
            # Write code here
            …

 # Driver Code
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

TRY

1. Take input operations as [ENQUEUE(A,B,C,D,E,F),DEQUEUE(),DEQUEUE(),DEQUEUE(),ENQUEUE(G,H,I] and verify the result.

2. Take input operations as [DEQUEUE(),ENQUEUE(A,B,C,D,E,F),DEQUEUE(),ENQUEUE(G,H,I] and verify the result.

5. Graph Representation

5.1 Build a graph

You are given an integer n. Determine if there is an unconnected graph with n vertices that contains at least two connected components and contains the number of edges that is equal to the number of vertices. Each vertex must follow one of these conditions:

- Its degree is less than or equal to 1.
- It's a cut-vertex.

Note:

- The graph must be simple.
- Loops and multiple edges are not allowed.

Input: First line: n

Output: Print Yes if it is an unconnected graph. Otherwise, print No.

Sample Input	Sample Output
3	No

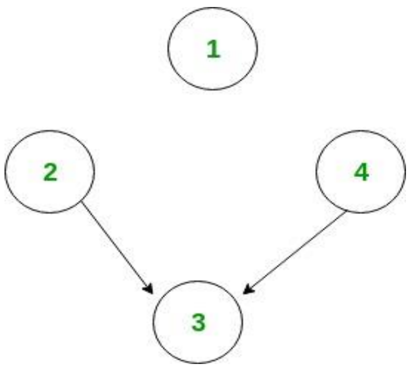
Constraints: $1 \leq n \leq 100$

Explanation: There is only one graph with the number of edges equal to the number of vertices (triangle) which is connected.

5.2 Number of Sink Nodes in a Graph

Given a Directed Acyclic Graph of n nodes (numbered from 1 to n) and m edges. The task is to find the number of sink nodes. A sink node is a node such that no edge emerges out of it.

Input: n = 4, m = 2, edges[] = {{2, 3}, {4, 3}}



Only node 1 and node 3 are sink nodes.

Input: n = 4, m = 2, edges[] = {{3, 2}, {3, 4}}

Output: 3

The idea is to iterate through all the edges. And for each edge, mark the source node from which the edge emerged out. Now, for each node check if it is marked or not. And count the unmarked nodes.

Algorithm:

1. Make any array A[] of size equal to the number of nodes and initialize to 1.

2. Traverse all the edges one by one, say, u -> v.

   (i) Mark A[u] as 1.

3. Now traverse whole array A[] and count number of unmarked nodes.

Hints:

```
# Program to count number if sink nodes

# Return the number of Sink Nodes.
def countSink(n, m, edgeFrom, edgeTo):
    # Write code here
            …

   return count

# Driver Code
n = 4
m = 2
edgeFrom = [2, 4]
edgeTo = [3, 3]

print(countSink(n, m, edgeFrom, edgeTo))
```

5.3 Connected Components in a Graph

Given n, i.e. total number of nodes in an undirected graph numbered from 1 to n and an integer e, i.e. total number of edges in the graph. Calculate the total number of connected components in the graph. A connected component is a set of vertices in a graph that are linked to each other by paths.

Input: First line of input line contains two integers' n and e. Next e line will contain two integers u and v meaning that node u and node v are connected to each other in undirected fashion.

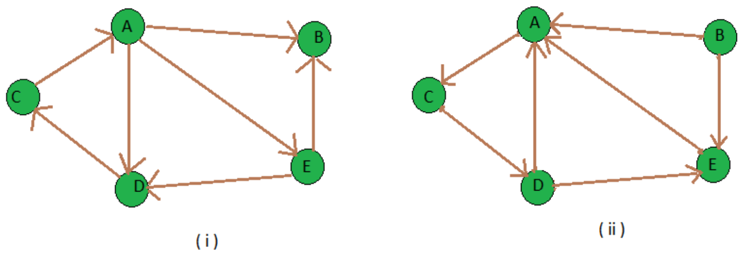
Output: For each input graph print an integer x denoting total number of connected components.

Sample Input	Sample Output
8 5	3
1 2	
2 3	
2 4	
3 5	
6 7	

Constraints: All the input values are well within the integer range.

5.4 Transpose Graph

Transpose of a directed graph G is another directed graph on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G. That is, if G contains an edge (u, v) then the converse/transpose/reverse of G contains an edge (v, u) and vice versa. Given a graph (represented as adjacency list), we need to find another graph which is the transpose of the given graph.



Input: figure (i) is the input graph.

Output: figure (ii) is the transpose graph of the given graph.

```
0--> 2
1--> 0  4
2--> 3
3--> 0  4
4--> 0
```

Explanation: We traverse the adjacency list and as we find a vertex v in the adjacency list of vertex u which indicates an edge from u to v in main graph, we just add an edge from v to u in the transpose graph i.e. add u in the adjacency list of vertex v of the new graph. Thus traversing lists of all vertices of main graph we can get the transpose graph.

Hints:

```
# find transpose of a graph.
# function to add an edge from vertex source to vertex dest
def addEdge(adj, src, dest):
    adj[src].append(dest)

# function to print adjacency list of a graph
def displayGraph(adj, v):
     …
        print()

# function to get Transpose of a graph taking adjacency list of given graph and that of Transpose graph
def transposeGraph(adj, transpose, v):
    # traverse the adjacency list of given graph and for each edge (u, v) add
    # an edge (v, u) in the transpose graph's adjacency list
    …

# Driver Code
v = 5
adj = [[] for i in range(v)]
addEdge(adj, 0, 1)
addEdge(adj, 0, 4)
addEdge(adj, 0, 3)
addEdge(adj, 2, 0)
addEdge(adj, 3, 2)
addEdge(adj, 4, 1)
addEdge(adj, 4, 3)

# Finding transpose of graph represented by adjacency list adj[]
transpose = [[]for i in range(v)]
transposeGraph(adj, transpose, v)

# Displaying adjacency list of transpose graph i.e. b
```

```
displayGraph(transpose, v)
```

TRY

1. Take input operations as addEdge(A, B), addEdge(A, D), addEdge(A, C), addEdge(C, A),addEdge(A, D), addEdge(C, B), addEdge(B, C) and verify the result.

5.5 Counting Triplets

You are given an undirected, complete graph G that contains N vertices. Each edge is colored in either white or black. You are required to determine the number of triplets (i, j, k) $(1 \le i < j < k \le N)$ of vertices such that the edges (i, j), (j, k), (i, k) are of the same color.

There are M white edges and (N (N-1)/2) – M black edges.

Input:

First line: Two integers – N and M $(3 \le N \le 10^5, 1 \le M \le 3 * 10^5)$

(i+1)th line: Two integers – u_i and v_i $(1 \le u_i, v_i \le N)$ denoting that the edge (u_i, v_i) is white in color.

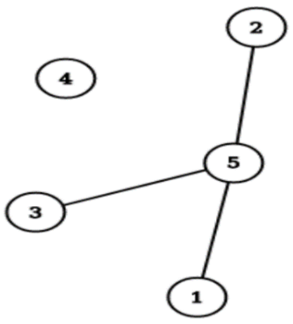
Note: The conditions $(u_i, v_i) \neq (u_j, v_j)$ and $(u_i, v_i) \neq (v_j, u_j)$ are satisfied for all $1 \le i < j \le M$.

.**Output:** Print an integer that denotes the number of triples that satisfy the mentioned condition.

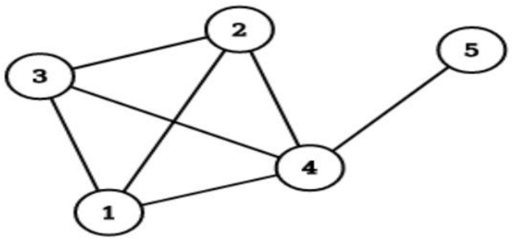
Sample Input	Sample Output
5 3	4
1 5	
2 5	
3 5	

Explanation: The triplets are: {(1, 2, 3), (1, 2, 4), (2, 3, 4), (1, 3, 4)}

The graph consisting of only white edges:



The graph consisting of only black edges:

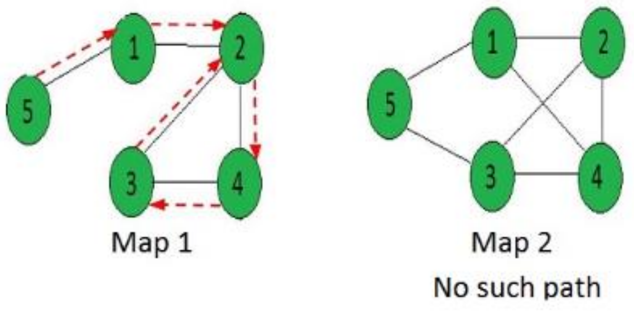


6. Graph Routing Algorithms

6.1 Seven Bridges of Konigsberg

There was 7 bridges connecting 4 lands around the city of Königsberg in Prussia. Was there any way to start from any of the land and go through each of the bridges once and only once? Euler first introduced graph theory to solve this problem. He considered each of the lands as a node of a graph and each bridge in between as an edge in between. Now he calculated if there is any Eulerian Path in that graph. If there is an Eulerian path then there is a solution otherwise not.

There are n nodes and m bridges in between these nodes. Print the possible path through each node using each edges (if possible), traveling through each edges only once.



Input: [[0, 1, 0, 0, 1],
[1, 0, 1, 1, 0],
[0, 1, 0, 1, 0],
[0, 1, 1, 0, 0],
[1, 0, 0, 0, 0]]

Output: 5 -> 1 -> 2 -> 4 -> 3 -> 2

Input: [[0, 1, 0, 1, 1],
[1, 0, 1, 0, 1],
[0, 1, 0, 1, 1],
[1, 1, 1, 0, 0],
[1, 0, 1, 0, 0]]

Output: "No Solution"

Hints:

```
# A Python program to print Eulerian trail in a
# given Eulerian or Semi-Eulerian Graph
from collections import defaultdict

class Graph:
# Constructor and destructor
def __init__(self, V):
self.V = V
 self.adj = defaultdict(list)

# functions to add and remove edge
```

```
def addEdge(self, u, v):
def rmvEdge(self, u, v):
   …
# Methods to print Eulerian tour
def printEulerTour(self):
 # Find a vertex with odd degree
  …
 # Print tour starting from oddv
self.printEulerUtil(u)
 print()
def printEulerUtil(self, u):
 # Recur for all the vertices adjacent to this vertex
for v in self.adj[u]:
 # If edge u-v is not removed and it's a valid next edge
# The function to check if edge u-v can be considered as next edge in Euler Tout
 def isValidNextEdge(self, u, v):
    # The edge u-v is valid in one of the following two cases:

    # 1) If v is the only adjacent vertex of u
        …

    # 2) If there are multiple adjacents, then u-v is not a bridge
    # Do following steps to check if u-v is a bridge
    # 2.a) count of vertices reachable from u
       …

    # 2.b) Remove edge (u, v) and after removing
    # the edge, count vertices reachable from u
        …

    # 2.c) Add the edge back to the graph
        self.addEdge(u, v)

    # 2.d) If count1 is greater, then edge (u, v) is a bridge
        return False if count1 > count2 else True

    # A DFS based function to count reachable vertices from v

    def DFSCount(self, v, visited):
        # Mark the current node as visited
        …
        # Recur for all the vertices adjacent to this vertex
        …
    # utility function to form edge between two vertices source and dest
    def makeEdge(src, dest):
        graph.addEdge(src, dest)

# Driver code
# Let us first create and test graphs shown in above figure
g1 = Graph(4)
g1.addEdge(0, 1)
g1.addEdge(0, 2)
g1.addEdge(1, 2)
g1.addEdge(2, 3)
g1.printEulerTour()

g3 = Graph(4)
g3.addEdge(0, 1)
```

```
g3.addEdge(1, 0)
g3.addEdge(0, 2)
g3.addEdge(2, 0)
g3.addEdge(2, 3)
g3.addEdge(3, 1)
g3.printEulerTour()
```

TRY

1. Take input: [[1, 0, 1, 0, 1], [1, 0, 1, 0, 0], [1, 1, 0, 1, 0], [0, 0, 1, 0, 0], [1, 0, 1, 0, 0]] and verify the result.
2. Take input: [[0, 0, 1, 0, 1], [0, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 1, 0, 0], [1, 1, 1, 0, 0]] and verify the result.

6.2 Hamiltonian Cycle

The Hamiltonian cycle of undirected graph G = <V, E> is the cycle containing each vertex in V. If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian.

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian path. Consider a graph G, and determine whether a given graph contains Hamiltonian cycle or not. If it contains, then prints the path. Following are the input and output of the required function.

Input: A 2D array graph [V][V] where V is the number of vertices in graph and graph [V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

Output: An array path [V] that should contain the Hamiltonian Path. Path [i] should represent the ith vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}.

```
(0)--(1)--(2)
 |   / \   |
 |  /   \  |
 | /     \ |
(3)-------(4)
```

And the following graph doesn't contain any Hamiltonian Cycle.

```
(0)--(1)--(2)
 |   / \   |
 |  /   \  |
 | /     \ |
(3)       (4)
```

Backtracking Algorithm: Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

Hints:

```
# Python program for solution of Hamiltonian cycle problem
```

```
class Graph():
    def __init__(self, vertices):
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]
        self.V = vertices

    def isSafe(self, v, pos, path):
        # Check if current vertex and last vertex in path are adjacent
      …

    # A recursive utility function to solve Hamiltonian cycle problem
    def hamCycleUtil(self, path, pos):
        …

    def hamCycle(self):
        …

    def printSolution(self, path):
        print ("Solution Exists: Following","is one Hamiltonian Cycle")
        for vertex in path:
            print (vertex, end = " ")
        print (path[0], "\n")

# Driver Code

''' Let us create the following graph
    (0)--(1)--(2)
    |   / \   |
    |  /   \  |
    | /     \ |
    (3)-------(4) '''
g1 = Graph(5)
g1.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
            [0, 1, 0, 0, 1,],[1, 1, 0, 0, 1],
            [0, 1, 1, 1, 0], ]

# Print the solution
g1.hamCycle();

''' Let us create the following graph
    (0)--(1)--(2)
    |   / \   |
    |  /   \  |
    | /     \ |
    (3)     (4) '''
g2 = Graph(5)
g2.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
        [0, 1, 0, 0, 1,], [1, 1, 0, 0, 0],
        [0, 1, 1, 0, 0], ]

# Print the solution
g2.hamCycle();
```

TRY

1. Take a graph = [[1, 1, 0, 1, 0], [1, 1, 1, 1, 1], [0, 1, 0, 1, 1,], [1, 1, 0, 1, 0], [0, 1, 1, 1, 0],] and verify the results.

6.3 Number of Hamiltonian Cycle

Given an undirected complete graph of N vertices where N > 2. The task is to find the number of different Hamiltonian cycle of the graph.

Complete Graph: A graph is said to be complete if each possible vertices is connected through an Edge.

Hamiltonian Cycle: It is a closed walk such that each vertex is visited at most once except the initial vertex. and it is not necessary to visit all the edges.

Formula: (N – 1)! / 2

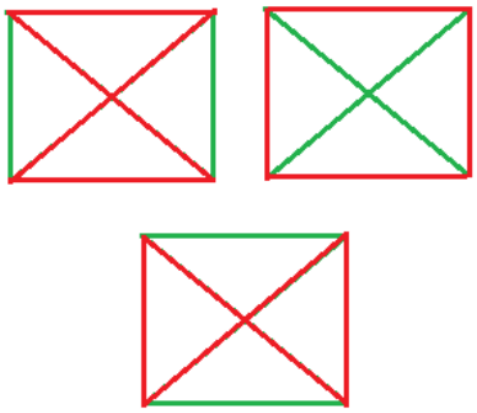
Input: N = 6

Output: Hamiltonian cycles = 60

Input: N = 4

Output: Hamiltonian cycles = 3

Explanation: Let us take the example of N = 4 complete undirected graph, The 3 different Hamiltonian cycle is as shown below:



Hints:

```
# Number of Hamiltonian cycles
import math as mt

# Function that calculates number of Hamiltonian cycle
def Cycles(N):
    …

# Driver code
N = 5
Number = Cycles(N)
print("Hamiltonian cycles = ", Number)
```

TRY

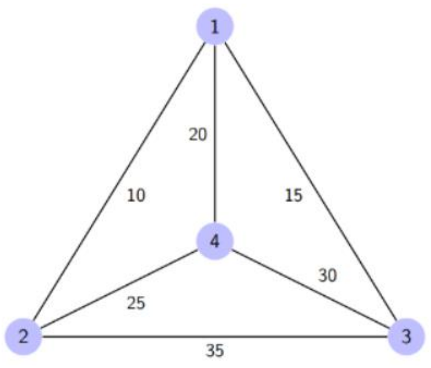
1. Take an input N=7 and verify the results.
2. Take an input N=10 and verify the results.

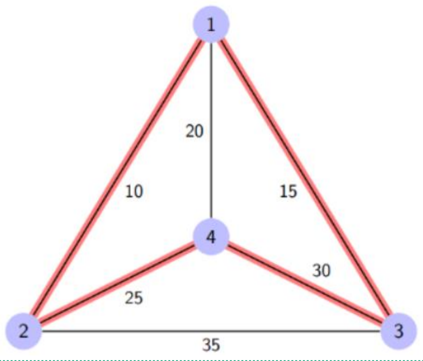
7. Shortest Path Algorithms

7.1 Travelling Salesman Problem

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. The problem statement gives a list of cities along with the distances between each city.

Objective: To start from the origin city, visit other cities only once, and return to the original city again. Our target is to find the shortest possible path to complete the round-trip route.





Here a graph is given where 1, 2, 3, and 4 represent the cities, and the weight associated with every edge represents the distance between those cities. The goal is to find the shortest possible path for the tour that starts from the origin city, traverses the graph while only visiting the other cities or nodes once, and returns to the origin city.
For the above graph, the optimal route is to follow the minimum cost path: 1 – 2 – 4 – 3 - 1. And this shortest route would cost 10 + 25 + 30 + 15 =80

Algorithm for Traveling Salesman Problem: We will use the dynamic programming approach to solve the Travelling Salesman Problem (TSP).

- A graph G=(V, E), which is a set of vertices and edges.
- V is the set of vertices.
- E is the set of edges.
- Vertices are connected through edges.
- Dist(i,j) denotes the non-negative distance between two vertices, i and j.

Let's assume S is the subset of cities and belongs to {1, 2, 3, …, n} where 1, 2, 3…n are the cities and i, j are two cities in that subset. Now cost(i, S, j) is defined in such a way as the length of the shortest path visiting node in S, which is exactly once having the starting and ending point as i and j respectively.

For example, cost (1, {2, 3, 4}, 1) denotes the length of the shortest path where:

- Starting city is 1
- Cities 2, 3, and 4 are visited only once
- The ending point is 1

The dynamic programming algorithm would be:

- Set cost(i,, i) = 0, which means we start and end at i, and the cost is 0.
- When |S| > 1, we define cost(i, S, 1) = ∝ where i !=1 . Because initially, we do not know the exact cost to reach city i to city 1 through other cities.
- Now, we need to start at 1 and complete the tour. We need to select the next city in such a way-
- cost(i, S, j) = min cost (i, S−{i}, j) + dist(i, j) where i ∈ S and i ≠ j

For the given figure above, the adjacency matrix would be the following:

dist(i, j)	1	2	3	4
1	0	10	15	20
2	10	0	35	25
3	15	35	0	30
4	20	25	30	0

Now S = {1, 2, 3, 4}. There are four elements. Hence the number of subsets will be 2^4 or 16. Those subsets are-
1) |S| = Null:
{Φ}

2) |S| = 1:
{{1}, {2}, {3}, {4}}

3) |S| = 2:
{{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}

4) |S| = 3:
{{1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 3, 4}}

5) |S| = 4:
{{1, 2, 3, 4}}

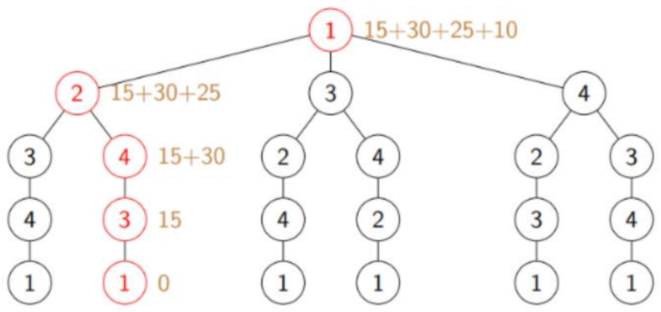
As we are starting at 1, we could discard the subsets containing city 1. The algorithm calculation steps:
1) |S| = Φ:
cost (2, Φ, 1) = dist(2, 1) = 10
cost (3, Φ, 1) = dist(3, 1) = 15
cost (4, Φ, 1) = dist(4, 1) = 20

2) |S| = 1:
cost (2, {3}, 1) = dist(2, 3) + cost (3, Φ, 1) = 35+15 = 50
cost (2, {4}, 1) = dist(2, 4) + cost (4, Φ, 1) = 25+20 = 45
cost (3, {2}, 1) = dist(3, 2) + cost (2, Φ, 1) = 35+10 = 45
cost (3, {4}, 1) = dist(3, 4) + cost (4, Φ, 1) = 30+20 = 50
cost (4, {2}, 1) = dist(4, 2) + cost (2, Φ, 1) = 25+10 = 35
cost (4, {3}, 1) = dist(4, 3) + cost (3, Φ, 1) = 30+15 = 45

3) |S| = 2:
cost (2, {3, 4}, 1) = min [dist[2,3] + Cost(3,{4},1) = 35+50 = 85,
dist[2,4]+Cost(4,{3},1) = 25+45 = 70] = 70
cost (3, {2, 4}, 1) = min [dist[3,2] + Cost(2,{4},1) = 35+45 = 80,
dist[3,4]+Cost(4,{2},1) = 30+35 = 65] = 65
cost (4, {2, 3}, 1) = min [dist[4,2] + Cost(2,{3},1) = 25+50 = 75
dist[4,3] + Cost(3,{2},1) = 30+45 = 75] = 75

4) |S| = 3:
cost (1, {2, 3, 4}, 1) = min [dist[1,2]+Cost(2,{3,4},1) = 10+70 = 80
dist[1,3]+Cost(3,{2,4},1) = 15+65 = 80
dist[1,4]+Cost(4,{2,3},1) = 20+75 = 95] = 80

So the optimal solution would be 1-2-4-3-1



Hints:

```
from sys import maxsize
from itertools, import permutations
V = 4
def tsp(graph, s):
                ...

# Driver code
graph = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]
s = 0
print(tsp(graph, s))
```

TRY

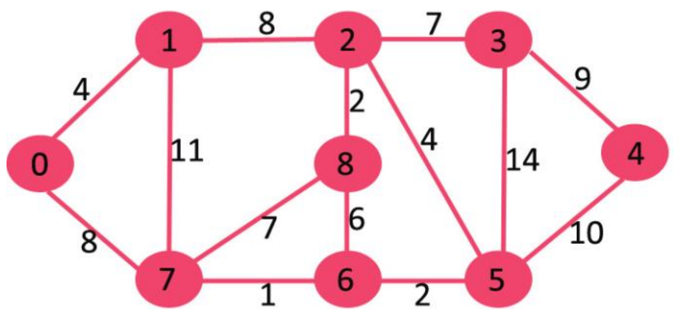
1. Take a below table values and verify the results.

dist(i, j)	1	2	3	4
1	0	40	25	40
2	20	0	35	25
3	25	35	0	60
4	40	25	30	0

7.2 Shortest Paths from Source to all Vertices (Dijkstra's Algorithm)

Given a graph and a source vertex in the graph, find the shortest paths from the source to all vertices in the given graph.

Input: src = 0, the graph is shown below.



Output: 0 4 12 19 21 11 9 8 14

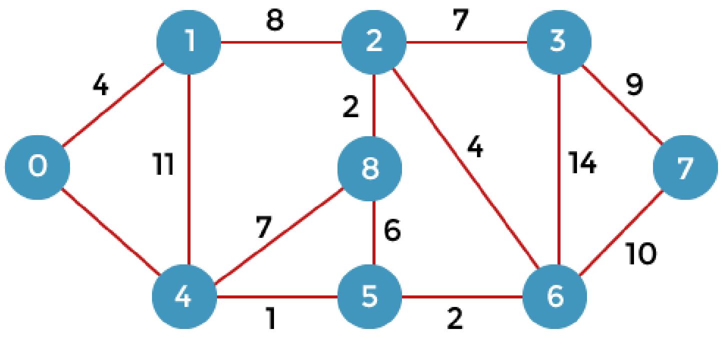
Explanation: The distance from 0 to 1 = 4.
The minimum distance from 0 to 2 = 12. 0->1->2
The minimum distance from 0 to 3 = 19. 0->1->2->3
The minimum distance from 0 to 4 = 21. 0->7->6->5->4
The minimum distance from 0 to 5 = 11. 0->7->6->5
The minimum distance from 0 to 6 = 9. 0->7->6
The minimum distance from 0 to 7 = 8. 0->7
The minimum distance from 0 to 8 = 14. 0->1->2->8

Hints:

```
# Dijkstra's single source shortest path algorithm. The program is for adjacency matrix representation of the graph

# Library for INT_MAX
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \tDistance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])

    # A utility function to find the vertex with minimum distance value,
    # from the set of vertices not yet included in shortest path tree
    def minDistance(self, dist, sptSet):
        …

    # Function that implements Dijkstra's single source shortest path
    # algorithm for a graph represented using adjacency matrix representation
    def dijkstra(self, src):
        …


# Driver's code
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
           [4, 0, 8, 0, 0, 0, 0, 11, 0],
           [0, 8, 0, 7, 0, 4, 0, 0, 2],
           [0, 0, 7, 0, 9, 14, 0, 0, 0],
           [0, 0, 0, 9, 0, 10, 0, 0, 0],
           [0, 0, 4, 14, 10, 0, 2, 0, 0],
           [0, 0, 0, 0, 0, 2, 0, 1, 6],
           [8, 11, 0, 0, 0, 0, 1, 0, 7],
           [0, 0, 2, 0, 0, 0, 6, 7, 0]
          ]

g.dijkstra(0)
```

TRY

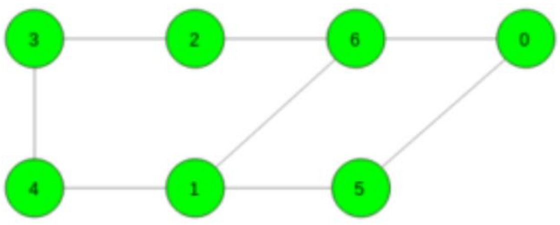
1. Take a below graph and verify the results.



7.3 Shortest Cycle in an Undirected Unweighted Graph

Given an undirected unweighted graph. The task is to find the length of the shortest cycle in the given graph. If no cycle exists print -1.

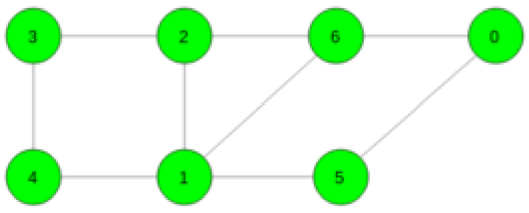
Input: Consider the graph given below



Output: 4

Cycle 6 -> 1 -> 5 -> 0 -> 6

Input: Consider the graph given below



Output: 3

Cycle 6 -> 1 -> 2 -> 6

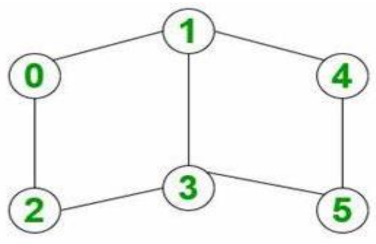
Hints:

```
from sys import maxsize as INT_MAX
from collections import deque

N = 100200

gr = [0] * N
for i in range(N):
    gr[i] = []
```

```
# Function to add edge
def add_edge(x: int, y: int) -> None:
    global gr
    gr[x].append(y)
    gr[y].append(x)

# Function to find the length of the shortest cycle in the graph
def shortest_cycle(n: int) -> int:

    # To store length of the shortest cycle
    ans = INT_MAX

    # For all vertices
    # write code here
    …

    # If graph contains no cycle
    if ans == INT_MAX:
        return -1

    # If graph contains cycle
    else:
        return ans

# Driver Code
# Number of vertices
n = 7
# Add edges
add_edge(0, 6)
add_edge(0, 5)
add_edge(5, 1)
add_edge(1, 6)
add_edge(2, 6)
add_edge(2, 3)
add_edge(3, 4)
add_edge(4, 1)

# Function call
print(shortest_cycle(n))
```

TRY

1. Take a below graph and verify the results.



7.4 Count Unique and all Possible Paths in a M x N Matrix

Count unique paths: The problem is to count all unique possible paths from the top left to the bottom right of a M X N matrix with the constraints that from each cell you can either move only to the right or down.

Input: M = 2, N = 2

Output: 2

Explanation: There are two paths
(0, 0) -> (0, 1) -> (1, 1)
(0, 0) -> (1, 0) -> (1, 1)

Input: M = 2, N = 3

Output: 3

Explanation: There are three paths
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2)
(0, 0) -> (0, 1) -> (1, 1) -> (1, 2)
(0, 0) -> (1, 0) -> (1, 1) -> (1, 2)

Count all possible paths: We can recursively move to right and down from the start until we reach the destination and then add up all valid paths to get the answer.

Procedure:

- Create a recursive function with parameters as row and column index
- Call this recursive function for N-1 and M-1
- In the recursive function
    - If N == 1 or M == 1 then return 1
    - else call the recursive function with (N-1, M) and (N, M-1) and return the sum of this
- Print the answer

Hints:

```
# Python program to count all possible paths from top left to bottom right

# Function to return count of possible paths to reach cell at row number m and column number n from the topmost leftmost cell (cell at 1, 1)

def numberOfPaths(m, n):
   …

# Driver program to test above function
m = 3
n = 3
print(numberOfPaths(m, n))
```

TRY

1. Take input : M = 3, N = 2 and verify the results.
2. Take input : M = 2, N = 1 and verify the results.

8. Graph Coloring

8.1 Graph Coloring using Greedy Algorithm

Greedy algorithm is used to assign colors to the vertices of a graph. It doesn't guarantee to use minimum colors, but it guarantees an upper bound on the number of colors. The basic algorithm never uses more than d+1 colors where d is the maximum degree of a vertex in the given graph.

Basic Greedy Coloring Algorithm:
1. Color first vertex with first color.
2. Do following for remaining V-1 vertices.

a) Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Hints:

```
# Implement greedy algorithm for graph coloring

def addEdge(adj, v, w):
    adj[v].append(w)
    # Note: the graph is undirected
    adj[w].append(v)
    return adj

# Assigns colors (starting from 0) to all
# vertices and prints the assignment of colors
def greedyColoring(adj, V):
    …

# Driver Code
g1 = [[] for i in range(5)]
g1 = addEdge(g1, 0, 1)
g1 = addEdge(g1, 0, 2)
g1 = addEdge(g1, 1, 2)
g1 = addEdge(g1, 1, 3)
g1 = addEdge(g1, 2, 3)
g1 = addEdge(g1, 3, 4)
print("Coloring of graph 1 ")
greedyColoring(g1, 5)

g2 = [[] for i in range(5)]
g2 = addEdge(g2, 0, 1)
g2 = addEdge(g2, 0, 2)
g2 = addEdge(g2, 1, 2)
g2 = addEdge(g2, 1, 4)
g2 = addEdge(g2, 2, 4)
g2 = addEdge(g2, 4, 3)
print("\nColoring of graph 2")
greedyColoring(g2, 5)
```

Output:

```
Coloring of graph 1
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 1

Coloring of graph 2
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 3
```

8.2 Coloring a Cycle Graph

Given the number of vertices in a Cyclic Graph. The task is to determine the Number of colors required to color the graph so that no two adjacent vertices have the same color.

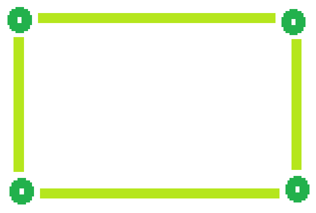
Approach:

- If the no. of vertices is Even then it is Even Cycle and to color such graph we require 2 colors.
- If the no. of vertices is Odd then it is Odd Cycle and to color such graph we require 3 colors.

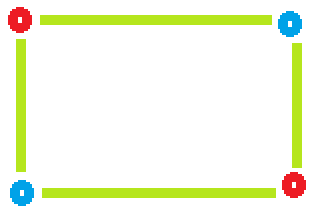
Input: Vertices = 3
Output: No. of colors require is: 3

Input: vertices = 4
Output: No. of colors require is: 2

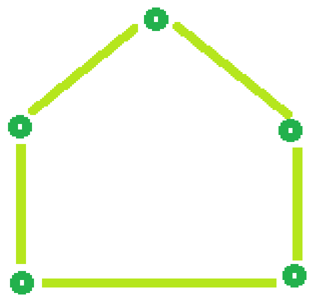
Example 1: Even Cycle: Number of vertices = 4



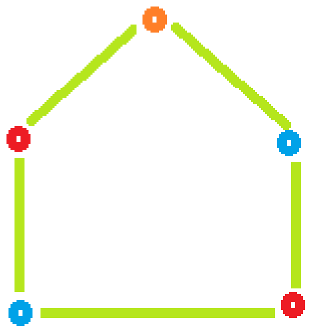
Color required = 2



Example 2: Odd Cycle: Number of vertices = 5



Color required = 3



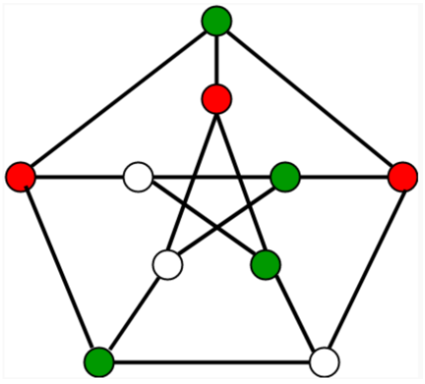
Hints:

```
# Find the number of colors required to color a cycle graph
# Function to find Color required.
def Color(vertices):
    …

# Driver Code
vertices = 3
print ("No. of colors require is:", Color(vertices))
```

8.3 m Coloring Problem

Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.

Note: Here coloring of a graph means the assignment of colors to all vertices
Following is an example of a graph that can be colored with 3 different colors:



Input: graph = {0, 1, 1, 1},
{1, 0, 1, 0},
{1, 1, 0, 1},
{1, 0, 1, 0}
Output: Solution Exists: Following are the assigned colors: 1 2 3 2
Explanation: By coloring the vertices with following colors, adjacent vertices does not have same colors

Input: graph = {1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1}
Output: Solution does not exist
Explanation: No solution exits

Generate all possible configurations of colors. Since each node can be colored using any of the m available colors, the total number of color configurations possible is mV. After generating a configuration of color, check if the adjacent vertices have the same color or not. If the conditions are met, print the combination and break the loop
Follow the given steps to solve the problem:

- Create a recursive function that takes the current index, number of vertices and output color array
- If the current index is equal to number of vertices. Check if the output color configuration is safe, i.e check if the adjacent vertices do not have same color. If the conditions are met, print the configuration and break
- Assign a color to a vertex (1 to m)
- For every assigned color recursively call the function with next index and number of vertices
- If any recursive function returns true break the loop and returns true.

Hints:

```
# Number of vertices in the graph
# define 4 4
# check if the colored graph is safe or not


def isSafe(graph, color):
    # check for every edge
    for i in range(4):
        for j in range(i + 1, 4):
            if (graph[i][j] and color[j] == color[i]):
                return False
    return True


def graphColoring(graph, m, i, color):
    # write your code here
    …

# /* A utility function to print solution */

def printSolution(color):
    print("Solution Exists:" " Following are the assigned colors ")
    for i in range(4):
        print(color[i], end=" ")


# Driver code
# /* Create following graph and test whether it is 3 colorable
# (3)---(2)
# |   / |
# |  /  |
# | /   |
# (0)---(1)
# */
graph = [
        [0, 1, 1, 1],
        [1, 0, 1, 0],
        [1, 1, 0, 1],
        [1, 0, 1, 0],
        ]
m = 3  # Number of colors

# Initialize all color values as 0.
# This initialization is needed
# correct functioning of isSafe()
color = [0 for i in range(4)]

# Function call
if (not graphColoring(graph, m, 0, color)):
        print("Solution does not exist")
```

8.4 Edge Coloring of a Graph

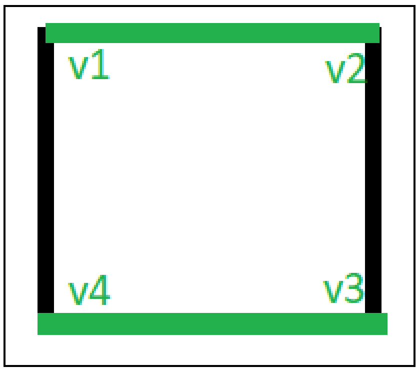
Edge coloring of a graph is an assignment of “colors” to the edges of the graph so that no two adjacent edges have the same color with an optimal number of colors. Two edges are said to be adjacent if they are connected to the same vertex. There is no known polynomial time algorithm for edge-coloring every graph with an optimal number of colors.

Input: u1 = 1, v1 = 4
u2 = 1, v2 = 2
u3 = 2, v3 = 3

u4 = 3, v4 = 4

Output: Edge 1 is of color 1
Edge 2 is of color 2
Edge 3 is of color 1
Edge 4 is of color 2

The above input shows the pair of vertices (u_i, v_i) who have an edge between them. The output shows the color assigned to the respective edges.



Edge colorings are one of several different types of graph coloring problems. The above figure of a Graph shows an edge coloring of a graph by the colors green and black, in which no adjacent edge have the same color.

Algorithm:
1. Use BFS traversal to start traversing the graph.
2. Pick any vertex and give different colors to all of the edges connected to it, and mark those edges as colored.
3. Traverse one of it's edges.
4. Repeat step to with a new vertex until all edges are colored.

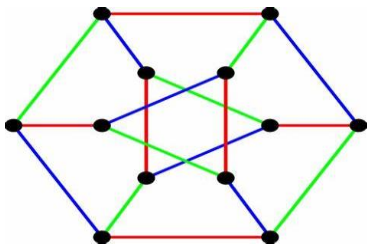
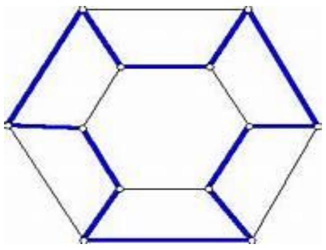
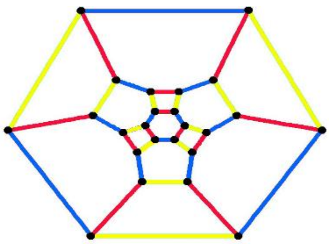
Hints:

```
# Edge Coloring

from queue import Queue

# function to determine the edge colors
def colorEdges(ptr, gra, edgeColors, isVisited):
        # Write your code here
        …


# Driver Function
empty=set()
# declaring vector of vector of pairs, to define Graph
gra=[]
edgeColors=[]
isVisited=[False]*100000

ver = 4
edge = 4
gra=[[] for _ in range(ver)]
edgeColors=[-1]*edge
gra[0].append((1, 0))
gra[1].append((0, 0))
gra[1].append((2, 1))
gra[2].append((1, 1))
```

```
gra[2].append((3, 2))
gra[3].append((2, 2))
gra[0].append((3, 3))
gra[3].append((0, 3))
colorEdges(0, gra, edgeColors, isVisited)

# printing all the edge colors
for i in range(edge):
    print("Edge {} is of color {}".format(i + 1,edgeColors[i] + 1))
```

TRY

1. Write a program to implement graph coloring and edge coloring by consider the below graph and verify the results.

9. Graph Traversal

9.1 Breadth First Search

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
For a given graph G, print BFS traversal from a given source vertex.

Hints:

```
# BFS traversal from a given source vertex.

from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):
            # Write your code here
            …


# Create a graph given in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
```

```
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal" " (starting from vertex 2)")
g.BFS(2)
```

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

9.2 Depth First Search

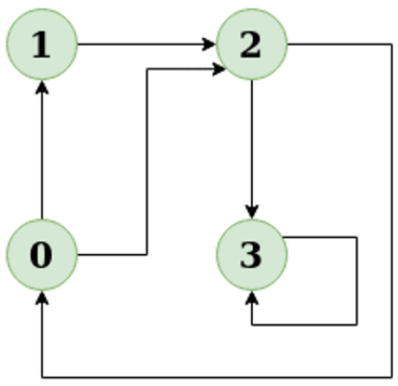
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

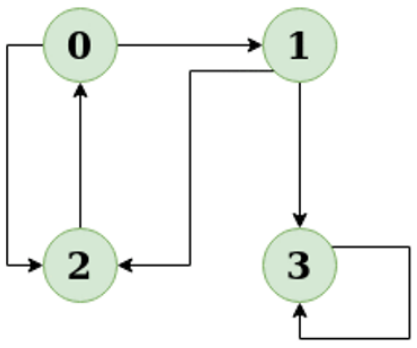
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:



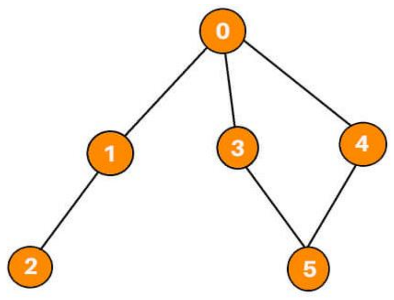
Hints:

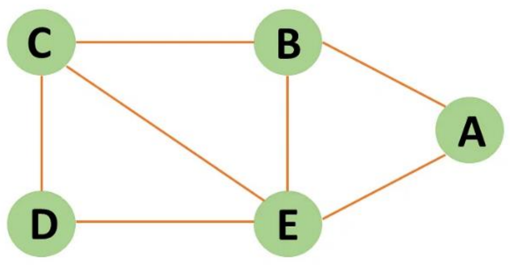
```
# DFS traversal from a given graph
from collections import defaultdict
```

```
# This class represents a directed graph using adjacency list representation
class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)


    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)


    # A function used by DFS
    def DFSUtil(self, v, visited):
        …


    # The function to do DFS traversal. It uses recursive DFSUtil()

    def DFS(self, v):
            # Write your code here
        …

# Driver's code
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print("Following is Depth First Traversal (starting from vertex 2)")
# Function call
g.DFS(2)
```

TRY

1. Write a program to implement breadth first search and depth first search by consider the below graph and verify the results.





10. Minimum Spanning Tree (MST)

10.1 Kruskal's Algorithm

In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

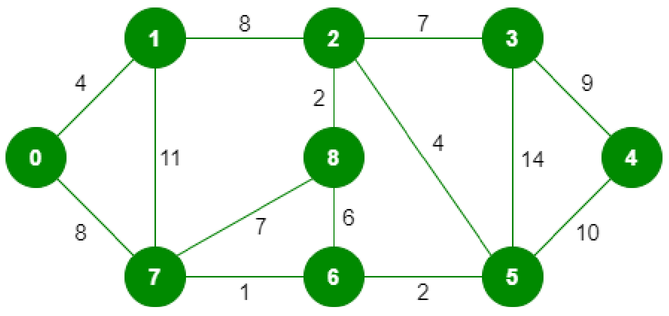
MST using Kruskal's algorithm:

1. Sort all the edges in non-decreasing order of their weight.

2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



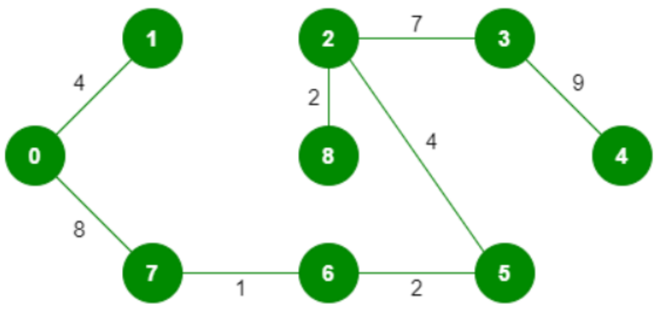
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:



Hints:

```
# Kruskal's algorithm to find minimum Spanning Tree of a given connected,
# undirected and weighted graph

# Class to represent a graph
```

```
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    # Function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        …

    def union(self, parent, rank, x, y):
        …

    def KruskalMST(self):
            # write your code here
            …

# Driver code
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

# Function call
g.KruskalMST()
```

Output: Following are the edges in the constructed MST

```
2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19
```

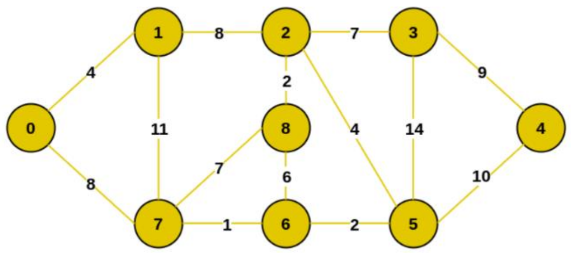
10.2 Prim’s Algorithm

The Prim’s algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

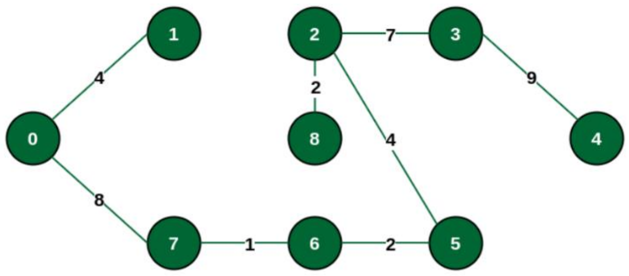
Prim’s Algorithm:
The working of Prim’s algorithm can be described by using the following steps:

1. Determine an arbitrary vertex as the starting vertex of the MST.
2. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
3. Find edges connecting any tree vertex with the fringe vertices.
4. Find the minimum among these edges.
5. Add the chosen edge to the MST if it does not form any cycle.
6. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



Hints:

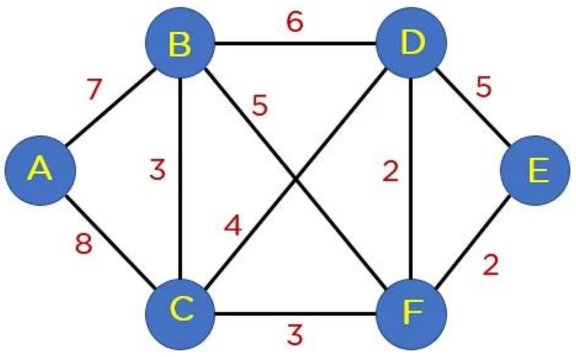
```
# Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph

# Library for INT_MAX
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # A utility function to print
    # the constructed MST stored in parent[]
    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        …
    def primMST(self):
        …


# Driver's code
g = Graph(5)
g.graph = [[0, 2, 0, 6, 0],
           [2, 0, 3, 8, 5],
           [0, 3, 0, 0, 7],
           [6, 8, 0, 0, 9],
           [0, 5, 7, 9, 0]]

g.primMST()
```

Output:

```
Edge    Weight

0 - 1      2

1 - 2      3

0 - 3      6

1 - 4      5
```

TRY

1. Write a program to implement kruskal's algorithm and prim's algorithm by consider the below graph and verify the results.



11. Roots of Equations

11.1 Bisection Method

The Bisection method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which f(x) = 0. The method is based on **The Intermediate Value Theorem** which states that if f(x) is a continuous function and there are two real numbers a and b such that f(a) * f(b) 0 and f(b) < 0), then it is guaranteed that it has at least one root between them.

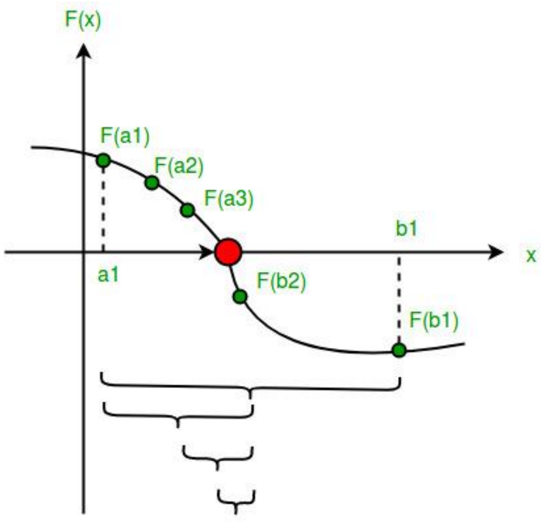
Assumptions:

1. f(x) is a continuous function in interval [a, b]
2. f(a) * f(b) < 0

Steps:

1. Find middle point c= (a + b)/2.
2. If f(c) == 0, then c is the root of the solution.
3. Else f(c) != 0
   - If value f(a)*f(c) < 0 then root lies between a and c. So we recur for a and c
   - Else If f(b)*f(c) < 0 then root lies between b and c. So we recur b and c.
   - Else given function doesn't follow one of assumptions.

Since root may be a floating point number, we repeat above steps while difference between a and b is greater than and equal to a value? (A very small value).



Hints:

```
# An example function whose solution is determined using Bisection Method.
# The function is x^3 - x^2  + 2
def func(x):
    return x*x*x - x*x + 2

# Prints root of func(x) with error of EPSILON
def bisection(a,b):
   # Write your code here
    …

# Driver code
# Initial values assumed
a =-200
b = 300
bisection (a, b)
```

Output: The value of root is : -1.0025

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.2 Method of False Position

Given a function f(x) on floating number x and two numbers 'a' and 'b' such that f(a)*f(b) < 0 and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

Input: A function of x, for example x3 – x2 + 2.
And two values: a = -200 and b = 300 such that
f(a)*f(b) < 0, i.e., f(a) and f(b) have opposite signs.

Output: The value of root is : -1.00
OR any other value close to root.

Hints:

```
MAX_ITER = 1000000

# An example function whose solution is determined using Regular Falsi Method.
# The function is x^3 - x^2 + 2
def func( x ):
    return (x * x * x - x * x + 2)

# Prints root of func(x) in interval [a, b]
def regulaFalsi( a , b):
    # Write your code here
    …

# Driver code to test above function
# Initial values assumed
a =-200
b = 300
regulaFalsi(a, b)
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.3 Newton Raphson Method

Given a function f(x) on floating number x and an initial guess for root, find root of function in interval. Here f(x) represents algebraic or transcendental equation.

Input: A function of x (for example $x^3 – x^2 + 2$), derivative function of x ($3x^2 – 2x$ for above example) and an initial guess x0 = -20

Output: The value of root is: -1.00 or any other value close to root.

Algorithm:
Input: initial x, func(x), derivFunc(x)
Output: Root of Func()

```
1.  Compute values of func(x) and derivFunc(x) for given initial x
2.  Compute h: h = func(x) / derivFunc(x)
3.  While h is greater than allowed error ε
      •  h = func(x) / derivFunc(x)
      •  x = x – h
```

Hints:

```
# Implementation of Newton Raphson Method for solving equations
# An example function whose solution is determined using Bisection Method.
# The function is x^3 - x^2 + 2
def func( x ):
    return x * x * x - x * x + 2

# Derivative of the above function which is 3*x^x - 2*x
def derivFunc( x ):
    return 3 * x * x - 2 * x

# Function to find the root
def newtonRaphson( x ):
    # Write your code here
    …

# Driver program
x0 = -20
newtonRaphson(x0)
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.4 Secant Method

The secant method is used to find the root of an equation f(x) = 0. It is started from two distinct estimates x1 and x2 for the root. It is an iterative procedure involving linear interpolation to a root. The iteration stops if the difference between two intermediate values is less than the convergence factor.

Input: Equation = x3 + x – 1
x1 = 0, x2 = 1, E = 0.0001

Output: Root of the given equation = 0.682326
No. of iteration=5

Algorithm

```
Initialize: x1, x2, E, n        // E = convergence indicator
calculate f(x1),f(x2)

if(f(x1) * f(x2) = E); //repeat the loop until the convergence
   print 'x0' //value of the root
   print 'n' //number of iteration
}
else
   print "cannot found a root in the given interval"
```

Hints:

```
# Find root of an equations using secant method
# Function takes value of x and returns f(x)
def f(x):
    # we are taking equation as x^3+x-1
    f = pow(x, 3) + x - 1;
    return f;

def secant(x1, x2, E):
   # Write your code here
    …

# Driver code
# initializing the values
x1 = 0;
x2 = 1;
E = 0.0001;
secant(x1, x2, E);
```

TRY

1. Take an input function x^2 - x + 2 and verify the results.
2. Take an input function x^3 - x^2 + 4 and verify the results.

11.5 Muller Method

Given a function f(x) on floating number x and three initial distinct guesses for root of the function, find the root of function. Here, f(x) can be an algebraic or transcendental function.

Input: A function f(x) = x   + 2x   + 10x - 20 and three initial guesses - 0, 1 and 2 .

Output: The value of the root is 1.3688 or any other value within permittable deviation from the root.

Input: A function f(x) = x   - 5x + 2 and three initial guesses - 0, 1 and 2.

Output: The value of the root is 0.4021 or any other value within permittable deviation from the root.

Hints:

```
# Find root of a function, f(x)
import math;

MAX_ITERATIONS = 10000;

# Function to calculate f(x)
def f(x):
    # Taking f(x) = x ^ 3 + 2x ^ 2 + 10x - 20
    return (1 * pow(x, 3) + 2 * x * x +
                        10 * x - 20);

def Muller(a, b, c):
    # Write your code here
    …


# Driver Code
a = 0;
b = 1;
c = 2;
Muller(a, b, c);
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

12. Numerical Integration

12.1 Trapezoidal Rule for Approximate Value of Definite Integral

Trapezoidal rule is used to find the approximation of a definite integral. The basic idea in Trapezoidal rule is to assume the region under the graph of the given function to be a trapezoid and calculate its area.

$$\int_a^b f(x)\,dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right]$$

Hints:

```
# Implement Trapezoidal rule

# A sample function whose definite integral's approximate value is
# computed using Trapezoidal rule
def y( x ):

    # Declaring the function
    # f(x) = 1/(1+x*x)
    return (1 / (1 + x * x))

# Function to evaluate the value of integral
def trapezoidal (a, b, n):
    # Write your code here
    …

# Driver code
# Range of definite integral
x0 = 0
xn = 1

# Number of grids. Higher value means more accuracy
n = 6
print ("Value of integral is ",
    "%.4f"%trapezoidal(x0, xn, n))
```

12.2 Simpson's 1/3 Rule

Simpson's 1/3 rule is a method for numerical approximation of definite integrals. Specifically, it is the following approximation:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6}\left(f(a) + 4f\frac{(a+b)}{2} + f(b)\right)$$

Procedure:

In order to integrate any function f(x) in the interval (a, b), follow the steps given below:

1. Select a value for n, which is the number of parts the interval is divided into.
2. Calculate the width, h = (b-a)/n
3. Calculate the values of x0 to xn as x0 = a, x1 = x0 + h, …..xn-1 = xn-2 + h, xn = b.
   Consider y = f(x). Now find the values of y (y0 to yn) for the corresponding x (x0 to xn) values.
4. Substitute all the above found values in the Simpson's Rule Formula to calculate the integral value.

Approximate value of the integral can be given by **Simpson's Rule:**

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f_0 + f_n + 4 * \sum_{i=1,3,5}^{n-1} f_i + 2 * \sum_{i=2,4,6}^{n-2} f_i\right)$$

Input: Evaluate logx dx within limit 4 to 5.2.

First we will divide interval into six equal parts as number of interval should be even.

x	: 4	4.2	4.4	4.6	4.8	5.0	5.2
$\log_x$	: 1.38	1.43	1.48	1.52	1.56	1.60	1.64

Output: Now we can calculate approximate value of integral using above formula:

= h/3[(1.38 + 1.64) + 4 * (1.43 + 1.52 + 1.60) +2 *(1.48 + 1.56)]

= 1.84

Hence the approximation of above integral is

1.827 using Simpson's 1/3 rule.

Hints:

```
# Simpson's 1 / 3 rule
import math

# Function to calculate f(x)
def func(x):
    return math.log(x)

# Function for approximate integral
def simpsons_(ll, ul, n):
    # Write your code here
     …

# Driver code
lower_limit = 4      # Lower limit
upper_limit = 5.2   # Upper limit
```

```
n = 6            # Number of interval
print("%.6f"% simpsons_(lower_limit, upper_limit, n))
```

12.3 Simpson's 3/8 Rule

The Simpson's 3/8 rule was developed by Thomas Simpson. This method is used for performing numerical integrations. This method is generally used for numerical approximation of definite integrals. Here, parabolas are used to approximate each part of curve.

Input: lower_limit = 1, upper_limit = 10, interval_limit = 10

Output: integration_result = 0.687927

Input: lower_limit = 1, upper_limit = 5, interval_limit = 3

Output: integration_result = 0.605835

Hints:

```
# Implement Simpson's 3/8 rule

# Given function to be integrated
def func(x):
   return (float(1) / ( 1 + x * x ))

# Function to perform calculations
def calculate(lower_limit, upper_limit, interval_limit ):
    # Write your code here
    …



# driver function
interval_limit = 10
lower_limit = 1
upper_limit = 10

integral_res = calculate(lower_limit, upper_limit, interval_limit)

# rounding the final answer to 6 decimal places
print (round(integral_res, 6))
```

13. Ordinary Differential Equations

13.1 The Euler Method

Given a differential equation dy/dx = f(x, y) with initial condition y(x0) = y0. Find its approximate solution using Euler method.

Euler Method:
In mathematics and computational science, the Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value.

Consider a differential equation dy/dx = f(x, y) with initial condition y(x0) = y0
then a successive approximation of this equation can be given by:

y(n+1) = y(n) + h * f(x(n), y(n))
where h = (x(n) – x(0)) / n, h indicates step size. Choosing smaller values of h leads to more accurate results and more computation time.

Example:
Consider below differential equation dy/dx = (x + y + xy) with initial condition y(0) = 1 and step size h = 0.025. Find y(0.1).

Solution:
f(x, y) = (x + y + xy)
x0 = 0, y0 = 1, h = 0.025
Now we can calculate y1 using Euler formula
y1 = y0 + h * f(x0, y0)
y1 = 1 + 0.025 *(0 + 1 + 0 * 1)
y1 = 1.025
y(0.025) = 1.025.
Similarly we can calculate y(0.050), y(0.075), ...y(0.1).
y(0.1) = 1.11167

Hints:

```
# Find approximation of an ordinary differential equation using Euler method.

# Consider a differential equation
# dy / dx =(x + y + xy)
def func( x, y ):
    return (x + y + x * y)

# Function for Euler formula
def euler( x0, y, h, x ):
    # write code here
    …

# Driver Code
# Initial Values
x0 = 0
y0 = 1
h = 0.025

# Value of x at which we need approximation
x = 0.1
euler(x0, y0, h, x)
```

13.2 Runge-Kutta Second Order Method

Given the following inputs:

1. An ordinary differential equation that defines the value of **dy/dx** in the form **x** and **y**.
$$\frac{dy}{dx} = f(x, y)$$
2. Initial value of y, i.e., **y(0)**.
$$y(0) = y_o$$

The task is to find the value of unknown function y at a given point x, i.e. **y(x)**.
Input: x0 = 0, y0 = 1, x = 2, h = 0.2

Output: y(x) = 0.645590

Input: x0 = 2, y0 = 1, x = 4, h = 0.4;

Output: y(x) = 4.122991

Approach:
The Runge-Kutta method finds an approximate value of y for a given x. Only first-order ordinary differential equations can be solved by using the Runge-Kutta 2nd-order method.
Below is the formula used to compute the next value yn+1 from the previous value yn. Therefore:
y_{n+1} = value of y at (x = n + 1)
y_n = value of y at (x = n)
where 0 ? n ? (x - x_0)/h, h is step height
$x_{n+1} = x_0 + h$

The essential formula to compute the value of y(n+1):

K1 = h * f(x, y)

K2 = h * f(x/2, y/2) or K1/2

$y_{n+1} = y_n + K^2 + (h^3)$

The formula basically computes the next value $\mathbf{y_{n+1}}$ using current $\mathbf{y_n}$ plus the weighted average of two increments:

- **K1** is the increment based on the slope at the beginning of the interval, using y.
- **K2** is the increment based on the slope at the midpoint of the interval, using **(y + h*K1/2)**.

Hints:

```
# Implement Runge-Kutta method

# A sample differential equation
# "dy/dx = (x - y)/2"

def dydx(x, y):
    return (x + y - 2)

# Finds value of y for a given x using step size h and initial value y0 at x0.
def rungeKutta(x0, y0, x, h):
    # write code here
    …

# Driver Code
x0 = 0
y = 1
x = 2
h = 0.2
print("y(x) =", rungeKutta(x0, y, x, h))
```

14. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

	Name of the contest	Minimum number of problems to solve	Required score
•	CodeChef	20	200

• Leetcode	20	200
• GeeksforGeeks	20	200
• SPOJ	5	50
• InterviewBit	10	1000
• Hackerrank	25	250
• Codeforces	10	100
• BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certification:

1. HackerRank - Problem Solving Skills Certification (Basic and Intermediate)
2. GeeksforGeeks – Data Structures and Algorithms Certification
3. CodeChef - Learn Python Certification
4. Interviewbit – DSA pro / Python pro
5. NPTEL – Programming, Data Structures and Algorithms
6. NPTEL – The Joy of Computing using Python

V. TEXT BOOKS:

1. Eric Matthes, "Python Crash Course: A Hands-On, Project-based Introduction to Programming", No Starch Press, 3rd Edition, 2023.
2. John M Zelle, "Python Programming: An Introduction to Computer Science", Ingram short title, 3rd Edition, 2016.

VI. REFERENCE BOOKS:

1. Yashavant Kanetkar, Aditya Kanetkar, "Let Us Python", BPB Publications, 2nd Edition, 2019.
2. Martin C. Brown, "Python: The Complete Reference", Mc. Graw Hill, Indian Edition, 2018.
3. Paul Barry, "Head First Python: A Brain-Friendly Guide", O'Reilly, 2nd Edition, 2016
4. Taneja Sheetal, Kumar Naveen, "Python Programming – A Modular Approach", Pearson, 1st Edition, 2017.
5. R Nageswar Rao, "Core Python Programming", Dreamtech Press, 2018.

VII. ELECTRONICS RESOURCES

8. https://realPython.com/Python3-object-oriented-programming/
9. https://Python.swaroopch.com/oop.html
10. https://Python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
11. https://www.programiz.com/Python-programming/
12. https://www.geeksforgeeks.org/python-programming-language/

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

ENGINEERING GRAPHICS								
I Semester: CSE / CSE(DS) / CSE(CS) **II Semester:** ECE / EEE / CSE (AI&ML) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AMED03	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	1	2	2	40	60	100
Contact Classes: 15	**Tutorial Classes: Nil**	**Practical Classes: 30**				**Total Classes:45**		
Prerequisite: Fundamentals of Geometry								

I. COURSE OVERVIEW:

Introduction to graphical representation using free hand drawing and computer-aided drafting. Engineering graphics covers basic engineering drawing techniques such as lines & lettering, geometrical constructions, principles of tangency, orthographic projections, sectional views, and dimensioning. This course assists to draw 2D drawings for industrial applications.

II. COURSES OBJECTIVES:

The students will try to learn

1. The basic engineering drawing formats.
2. Projections of points, lines, planes and solids at inclinations of horizontal plane and vertical plane.
3. Use of computer-aided design (CAD) to communicate concepts and ideas in the design of three-dimensional engineering products.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Demonstrate an ability to dimension and annotate two-dimensional engineering graphics.
CO2	Demonstrate the freehand sketching to aid in the visualization process and to efficiently communicate ideas graphically.
CO3	Make use of CAD software for the creation of 3D models and 2D engineering graphics.
CO4	Comprehend the principles and techniques for creating sectional views of three-dimensional solids in engineering graphics.
CO5	Explain the application of industry standards and best practices applied in engineering graphics.
CO6	Apply the general projection theory with emphasis on orthographic projection to represent three-dimensional objects in two-dimensional views

IV. COURE CONTENT:

EXCERCISES ON ENGINEERING GRAPHICS

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Introduction to AUTOCAD

AutoCAD is a widely-used computer-aided design (CAD) software application developed by Autodesk. It has been an industry standard for drafting and designing since its inception in the early 1980s. AutoCAD provides a versatile platform for creating and editing 2D and 3D drawings and models, making it an essential tool in various fields such as architecture, engineering, construction, manufacturing, and more.

i. Install AUTO CAD
ii. Purpose and Application
iii. Interface and Tools
iv. Precision and Accuracy
v. 2D and 3D Modeling
vi. Collaboration and Sharing
vii. Customization
viii. Industry Usage
ix. Versions and Licensing

1.1 Commands

The main purpose of using commands and shortcuts in AutoCAD boils down to increased productivity. They allow you to execute functions more quickly, as you don't need to search through the entire AutoCAD interface for the right tool. You can just type the command, and the function window appears.

i. Basic Drawing Commands
ii. Editing Commands
iii. Dimensioning Commands
iv. Advanced and Miscellaneous Commands

Basic Drawing Commands:
- Line (LINE): Draws straight line segments between two points.
- Circle (CIRCLE): Creates circles by specifying a center point and radius.
- Rectangle (RECTANGLE): Constructs rectangles by defining two opposing corners.
- Arc (ARC): Draws arcs based on different methods, such as specifying start, end, and radius or center, start, and angle.

Editing Commands:
- Erase (ERASE): Deletes selected objects from the drawing.
- Copy (COPY): Copies objects to a specified location.
- Move (MOVE): Relocates selected objects to a different position.
- Trim (TRIM): Cuts selected objects at the cutting edges defined by other objects.
- Extend (EXTEND): Extends objects to meet the boundaries of other objects.

Dimensioning Commands:
- Line Dimension (DIMLINEAR): Adds linear dimensions to objects.
- Aligned Dimension (DIMALIGNED): Creates dimensions aligned with an angle of the object.
- Radial Dimension (DIMRADIUS): Add radius dimensions to arcs and circles.
- Diameter Dimension (DIMDIAMETER): Creates diameter dimensions for circles.

Advanced and Miscellaneous Commands:
- Hatch (HATCH): Fills enclosed areas with a pattern or gradient.
- Offset (OFFSET): Creates parallel copies of objects at a specified distance.
- Block (BLOCK): Defines reusable blocks (collections of objects) in the drawing.
- Insert (INSERT): Inserts predefined blocks into the drawing.
- Viewport (VPORTS): Manages viewports for layout and plotting in paper space.

- Layer (LAYER): Manages layers for organizing and controlling object visibility.

TRY: Observe Exercise 1.1 in Solid works and in Creo software.

2. Introduction to Engineering Drawing

Engineering drawing, often referred to as technical drawing or drafting, is a graphical representation of an object, system, or structure used in various fields of engineering, manufacturing, and architecture. These drawings serve as a universal language that communicates design ideas, specifications, and instructions in a precise and standardized manner.

2.1 Basic Exercises

To be proficient in engineering drawing, basic exercises are required.

i. Identify the basic tools used for drafting
ii. Types of lines
iii. Arcs
iv. Circles

2.2 Practicing the standard lettering and numbering

Practicing standard lettering and numbering in engineering drawing is crucial for creating clear, professional, and easily understandable technical drawings. Proper lettering and numbering enhance communication and ensure that your drawings convey information accurately.

The following exercises are to be practiced to become proficient in lettering and numbering.

1. Use the correct fonts
2. Maintain uniformity
3. Lettering style
4. Height and spacing

Try: The following questions are to be answered in Solid works

1. How to use correct fonts in Solid Works
2. What are the commands are used to maintain uniformity of lettering and numbering.

3. Dimensioning

Dimensioning in engineering drawing is a crucial aspect that involves adding measurements and annotations to convey the size, location, and tolerances of objects, features, and components accurately. Proper dimensioning is essential for manufacturing, construction, and other engineering processes.

3.1 Exercises on Dimensioning

1. Understanding and use of the conventional dimensioning techniques.
2. Placing the dimension lines
3. Extension lines
4. Dimensions on angles

Hint: 1. The following Fig.3.1 shows the type of dimensioning on 2D drawing.

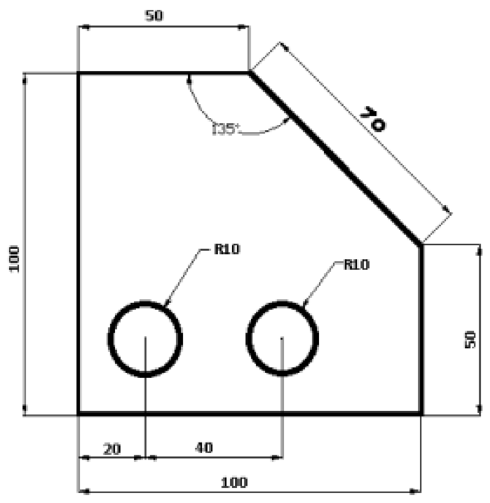


Fig.3.1 Dimensioning on 2D drawing

Hint: 2. The following Fig.3.2 shows the type of dimensioning on concentric circles.

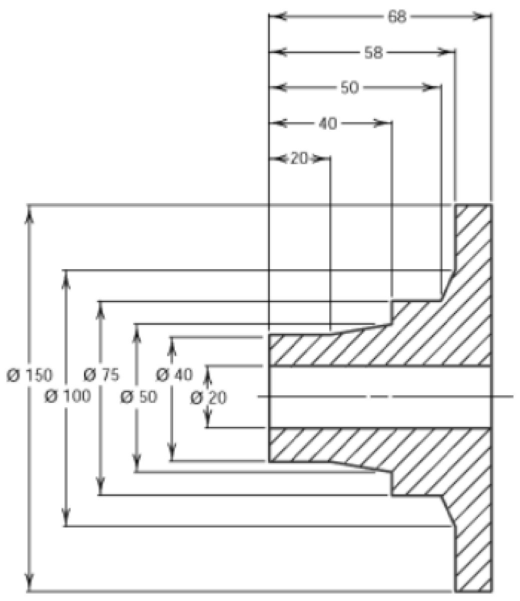


Fig.3.2 Dimensioning on Concentric circle

Try: Demonstrate the exercise 3 in Solid Works and CREO software.

4. Geometrical Constructions

Geometric construction is useful for learning how to use geometric tools like a ruler, compass, and straightedge to draw various angles, line segments, bisectors, and other forms of polygons, arcs, circles, and other geometric figures. Fig.4.1 shows the various geometric shapes to draw orthographic projections of lines, planes and solids.

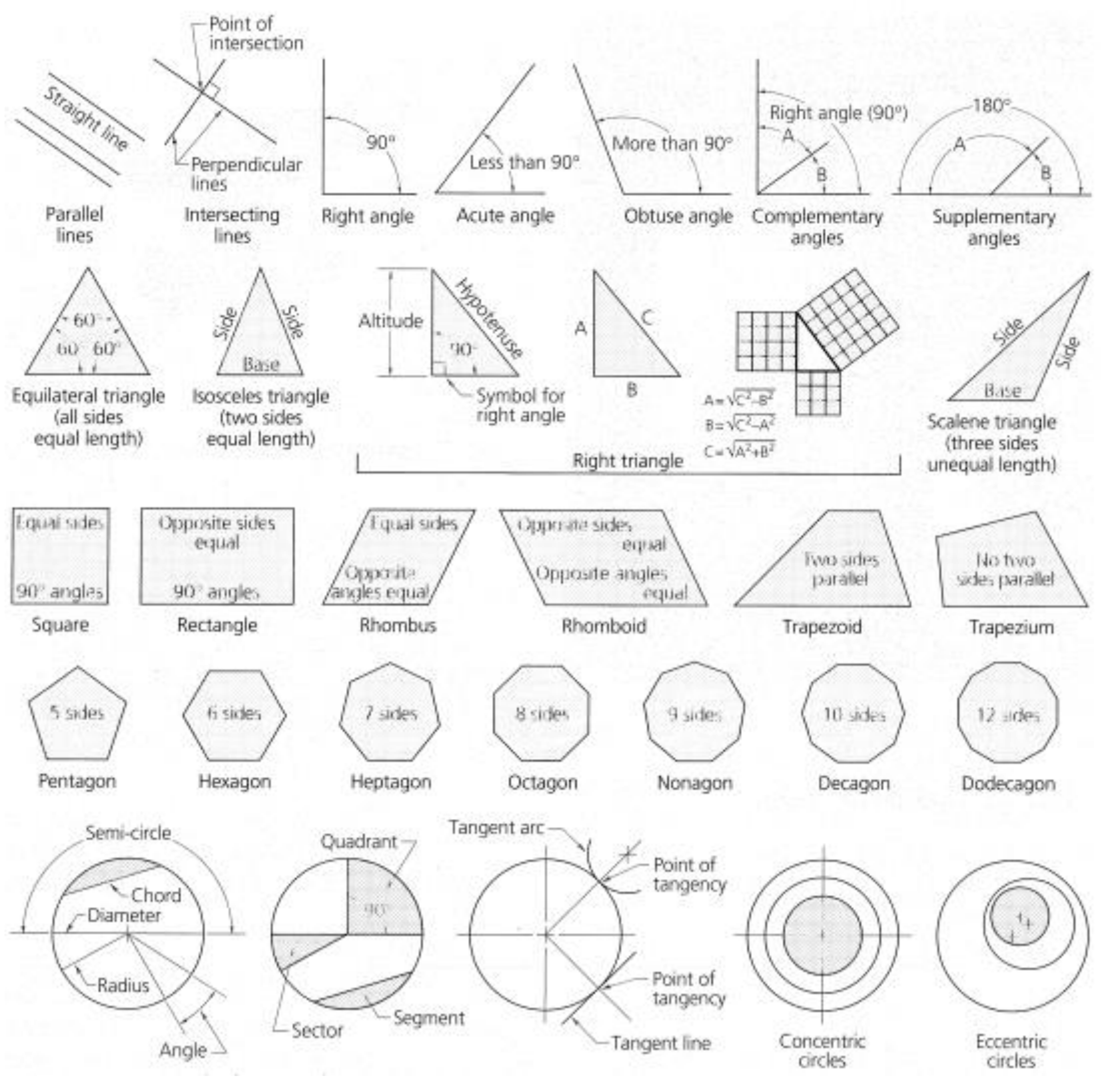


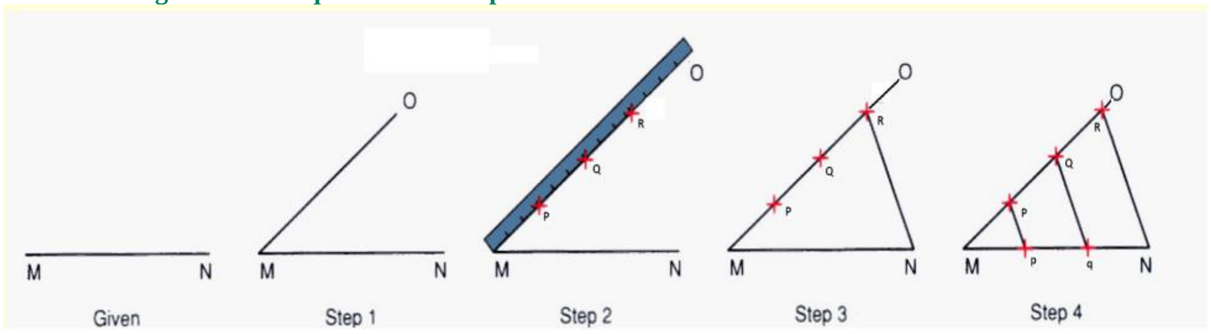
Fig.4.1 Geometric shapes

4.1. Exercises on Geometrical Constructions

To become proficient in engineering graphics geometrical constructions are required: Drawing lines, angles, triangle, square, pentagon, hexagon, octagon. Dividing line into equal or proportional parts. Drawing lines and arcs tangent to each other.

1. Divide a 16 cm straight line into a given number of equal parts say 5.
2. Divide a 8 cm line into 9 number of parts.
3. Bisect a given 45 degree sector.
4. Bisect a given straight line.
5. To draw a perpendicular to a given line from a point within it.
6. Construct a regular polygon, given the length of its side.

Hint: Dividing a line into equal number of parts



Try: The exercise 4 in Solid Works and CREO software.

5. Conic Sections

A conic section, conic or a quadratic curve is a curve obtained from a cone's surface intersecting a plane. The three types of conic section are the hyperbola, the parabola, and the ellipse.

5.1. Exercises on Conic Sections

1. Draw an ellipse with the distance of the focus from the directrix at 50mm and eccentricity = 2/3 (Eccentricity method)
2. Draw an ellipse with the distance of the focus from the directrix at 60mm and eccentricity = 2/3 (Eccentricity method)
3. Draw an ellipse with the distance of the focus from the directrix at 80mm and eccentricity = 2/3 (Eccentricity method)
4. Draw a parabola with the distance of the focus from the directrix at 50 mm (Eccentricity method).
5. Draw a parabola with the distance of the focus from the directrix at 40mm (Eccentricity method).
6. A vertex of a hyperbola is 60 mm from its focus. Draw two parts of the hyperbola; if the eccentricity is 3/2.
7. A vertex of a hyperbola is 50 mm from its focus. Draw two parts of the hyperbola; if the eccentricity is 1.5.

Try: The exercise 5 in Solid Works and CREO software.

6. Technical Sketching and Shape Description

6.1. Projections of planes and regular solids

1. Draw the projections of a regular pentagon of 30 mm side, having its surface inclined at 30° to the H.P. and a side parallel to the H.P. and inclined at an angle of 60° to the V.P.
2. Draw the projections of a regular hexagon of 40 mm side, having one of its sides in the H.P. and inclined at 60° to the V.P., and its surface making an angle of 45° with the H.P.
3. Draw the projections of a regular hexagon of 35 mm side, having one of its sides in the H.P. and inclined at 50° to the V.P., and its surface making an angle of 45° with the H.P.
4. Draw the projections of a regular pentagon of 40 mm side, having one of its sides in the H.P. and inclined at 30° to the V.P., and its surface making an angle of 45° with the H.P.

Try: The exercise 6.3 in Solid Works and CREO software.

7. Sectional views

A sectional view represents the part of an object remaining after a portion is assumed to have been cut and removed. The exposed cut surface is then indicated by section lines. Hidden features behind the cutting plane are omitted, unless required for dimensioning or for definition of the part.

7.1. Exercise on Sectional views of right regular solids, prism, cylinder, pyramid, cone.

1. A pentagonal pyramid, base 40 mm side and axis 60 mm long has its base horizontal and an

edge of the base parallel to the V.P. A horizontal section plane cuts it at a distance of 20 mm above the base. Draw its front view and sectional top view.

2. A hexagonal prism, side of base 40 mm and height 70 mm is resting on one of its corners on the H.P. with a longer edge containing that corner inclined at 40° to the H.P. and a rectangular face parallel to the V.P. Draw the front view and sectional top view of the cut prism when a horizontal section plane cuts the prism in two equal halves. Draw the front view and sectional top view of the cut prism.
3. A pentagonal pyramid, base 40 mm side and axis 70 mm long has one of its triangular faces in the V.P. and the edge of the base contained by that face makes an angle of 40° with the H.P. Draw its projections.
4. Draw the projections of a cone, base 50 mm diameter and axis 75 mm long, lying on a generator on the ground with the top view of the axis making an angle of 45° with the V.P.

Try: The exercise 7.1 and 7.2 in Solid Works and CREO software.

8. Development of surfaces

Knowledge of development is very useful in sheet metal work, construction of storage vessels, chemical vessels, boilers, and chimneys. Such vessels are manufactured from plates that are cut according to these developments and then properly bend into desired shaped.

8.1. Exercise on Basics of development of surfaces

1. Draw the development of the lateral surfaces of a right square prism of edge of base 30 mm and axis 50 mm long.
2. Draw the development of the complete surface of a cylindrical drum. Diameter is 40 mm and height 60 mm.

8.2. Exercise on Development of surfaces of Prisms

1. A hexagonal prism of base side 20 mm and height 45 mm is resting on one of its ends on the HP with two of its lateral faces parallel to the VP. It is cut by a plane perpendicular to the VP and inclined at 30° to the HP. The plane meets the axis at a distance of 20 mm above the base. Draw the development of the lateral surfaces of the lower portion of the prism.
2. A hexagonal prism, edge of base 20 mm and axis 50 mm long, rests with its base on HP such that one of its rectangular faces is parallel to VP. It is cut by a plane perpendicular to VP, inclined at 45° to HP and passing through the right corner of the top face of the prism. (i) Draw the sectional top view. (ii)Develop the lateral surfaces of the truncated prism.
3. A pentagonal prism, side of base 25 mm and altitude 50 mm, rests on its base on the HP such that an edge of the base is parallel to VP and nearer to the observer. It is cut by a plane inclined at 45° to HP, perpendicular to VP and passing through the center of the axis. (i) Draw the development of the complete surfaces of the truncated prism.
4. A pentagonal prism of side of base 30 mm and altitude 60 mm stands on its base on HP such that a vertical face is parallel to VP and away from observer. It is cut by a plane perpendicular to VP, inclined at an angle of 50° to HP and passing through the axis 35 mm above the base. Draw the development of the lower portion of the prism.

Try : The exercise 8.1 and 8.2 in Solid Works and CREO software.

9. Exercise on Development of surfaces-2

9.1. Exercise on Development of surfaces of cylinder and cone

1. Draw the development of the lateral surface of the lower portion of a cylinder of diameter 50 mm and axis 70 mm when sectioned by a plane inclined at 40° to HP and perpendicular to VP and bisecting axis.
2. A Cone of base diameter 60 mm and height 70 mm is resting on its base on HP. It is cut by a plane perpendicular to VP and inclined at 30° to HP. The plane bisects the axis of the cone. Draw the development of its lateral surface.

9.2. Exercise on Development of surfaces of pyramid

1. Draw the development of the lateral surfaces of a square pyramid, side of base 25 mm and height 50 mm, resting with its base on HP and an edge of the base parallel to VP.
2. A square pyramid of base side 25 mm and altitude 50 mm rests on it base on the HP with two

sides of the base parallel to the VP. It is cut by a plane bisecting the axis and inclined a 30° to the base. Draw the development of the lateral surfaces of the lower part of the cut pyramid.

3. A pentagonal pyramid side of base 30 mm and height 52 mm stands with its base on HP and an edge of the base is parallel to VP and nearer to it. It is cut by a plane perpendicular to VP, inclined at 40° to HP and passing through a point on the axis 32 mm above the base. Draw the sectional top view. Develop the lateral surface of the truncated pyramid.

Try: The exercise 9.1 and 9.2 in Solid Works and CREO software.

10. Orthographic views

Orthographic views are two-dimensional views of three-dimensional objects. Orthographic views are created by projecting a view of an object onto a plane which is usually positioned so that it is parallel to one of the planes of the object.

10.1. Exercise on Conversion of isometric view to orthographic projections using CAD

1. Draw the front view, side view and top view for the below Fig.10.1

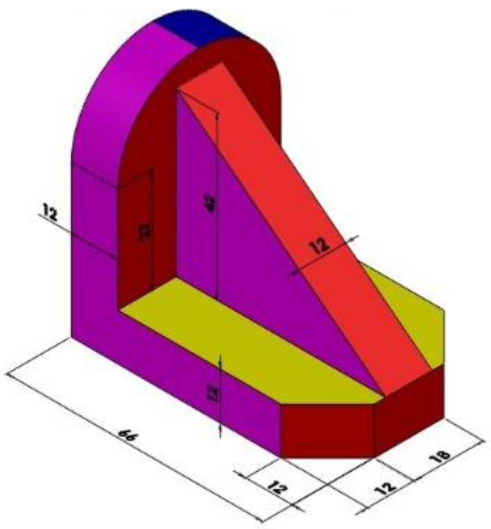


Fig.10.1

2. Draw the front view, side view and top view for the below Fig.10.2.

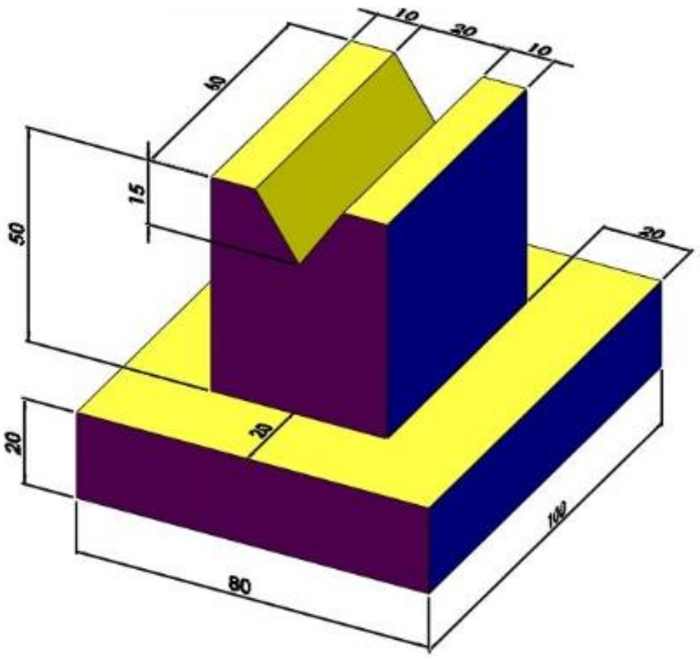


Fig.10.2

3. Draw the front view, side view and top view for the below Fig.10.3.

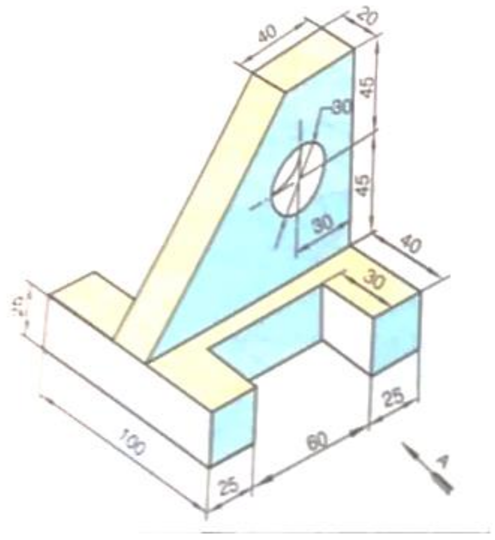


Fig.10.3

4. Draw the front view, side view and top view for the below Fig.10.4.

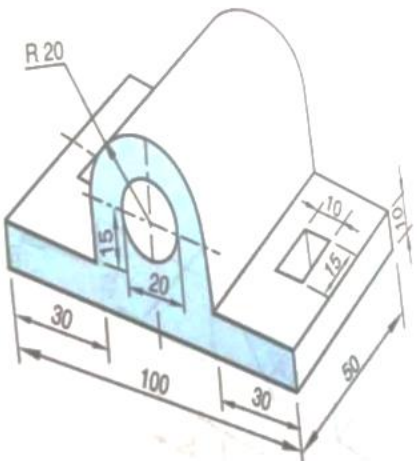


Fig.10.4

Try: Practice Exercise 10 in Solid Works

11. Isometric projection of planes

Isometric projection is a method for visually representing three-dimensional objects in two dimensions in technical and engineering drawings. It is an axonometric projection in which the three coordinate axes appear equally foreshortened and the angle between any two of them is 120 degrees.

11.1. Isometric scale

In engineering and technical drawing, an isometric scale is a method used to create an isometric projection of a three-dimensional object onto a two-dimensional surface, such as a drawing sheet or a computer screen. Isometric projection is a type of pictorial representation that shows an object in a three-dimensional view with all three principal axes (x, y, and z) at equal angles to the picture plane. An isometric scale is used to ensure that the dimensions and proportions of objects in the isometric drawing are accurate and maintain the proper relationships.

11.2. Exercise on Isometric projections of circle, square and rectangle and solids

1. Draw the isometric view of a circle of 60 mm diameter whose surface is parallel to the V.P.
2. Draw the isometric view of a square of side 60 mm whose surface is parallel to the H.P.
3. Draw the isometric view of a circle of 40 mm radius whose surface is parallel to the H.P.
4. Draw the isometric view of a square prism, side of the base 20 mm long and the axis 40 mm long, when its axis is i) Veritcal and ii) Horizontal.
5. Draw the isometric view of the semi-circle whose front view of its surface is parallel to the V.P.. The diameter of semi-circle is 60 mm.

Try : The exercise 11.2 in Solid Works and CREO software.

1. Isometric projections of solids

12.1. Exercise on conversion of orthographic view to isometric view using CAD

1. Draw the isometric view for the given orthographic views Fig.12.1

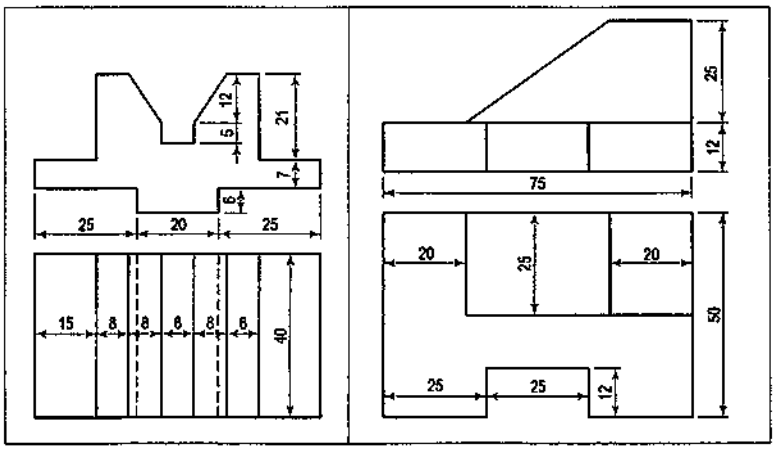


Fig.12.1

2. Draw the isometric view for the given orthographic views for Fig.12.2

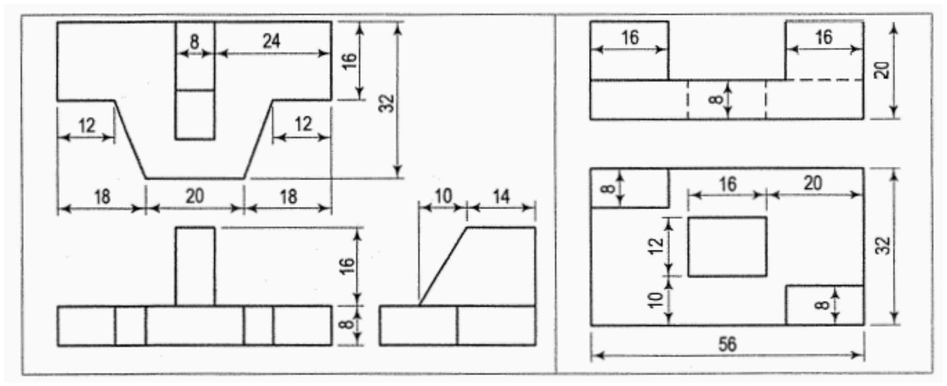


Fig.12.2

3.Draw the isometric view for the given orthographic views by assuming the dimensions for Fig.12.3

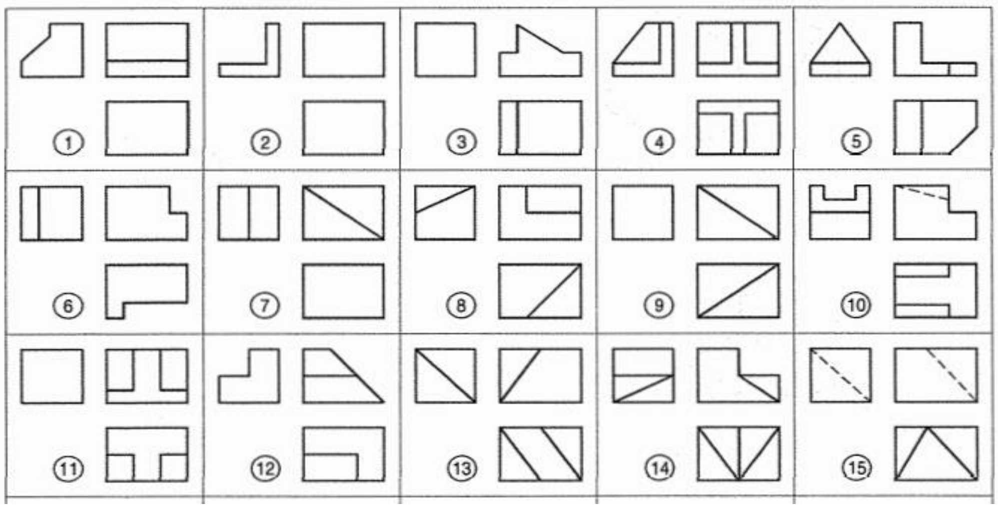


Fig.12.3

Try : The exercise 12 in Solid Works and CREO software.

13.Demonstration of SOLID WORKS Software

1. Introduction to SOLID WORKS
2. Demonstration of commands
3. 2D drawings

3. 3D drawings

14.Demonstration of CREO Software

1. Introduction to SOLID WORKS
2. Demonstration of commands
3. 2D drawings
3. 3D drawings

V. TEXT BOOKS:

1. Frederick E Giesecke, Alva Mitchell, Henry C Spencer, Ivan L Hill, John T Dygdon, James E. Novak, R. O. Loving, Shawna Lockhart, Cindy Johnson, *Technical Drawing with Engineering Graphics*, Pearson Education, 15th Edition, 2016.
2. Kulkarni D.M, Rastogi A.P. and Sarkar A.K., *Engineering Graphics with Auto CAD*. (Revised Edition), Prentice Hall India, New Delhi, 2011.
3. Donald Hearn, "*Computer Graphics*", 12th Edition, Pearson, 2021.

VI. REFERENCE BOOKS:

1. Basant Agrawal and C M Agrawal, *Engineering Drawing*, McGraw Hill, 3rd Edition, 2018.
2. James M. Leake, Molly Hathaway Goldstein, Jacob L. Borgerson, *Engineering Design Graphics, Modelling and Visualization*, Wiley, 3rd Edition, 2020.

VII. ELECTRONICS RESOURCES:

1. https://archive.nptel.ac.in/courses/112/103/112103019.
2. https://archive.nptel.ac.in/courses/112/105/112105294.

VIII. MATERIALS ONLINE:

3. Course Template
4. Laboratory manual

COURSE CONTENT

MOBILE APPLICATIONS DEVELOPMENT								
I Semester: Common for CSE / CSE(DS) / CSE(CS) **II Semester:** Common for CSE (AI&ML) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACSD04	**Skill**	0	0	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Object Oriented Programming								

I. COURSE OVERVIEW:

This course focuses on hands-on experience in designing, developing, and testing mobile applications for various platforms. It helps to gain practical skills in mobile app development, including user interface design, memory management, input methods, data handling, network techniques and URL loading. Students will be able to undertake engineering challenges such as designing and developing mobile applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The mobile application development for different platforms using appropriate tools and frameworks.

II. The user interface design with best practices for usability and user experience.

III. The process of debugging and troubleshooting for common issues in mobile app development.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Apply layout management and multi layout techniques to create adaptable user interface.
CO 2	Develop user interface for mobile application using widgets with event handling.
CO 3	Design push notifications for incoming messages.
CO 4	Create mobile application models using appropriate range of methods provided.
CO 5	Evaluate applications on mobile platforms with different configurations.
CO 6	Deploy applications to the android marketplace for distribution to app store.

IV. COURSE CONTENT:

GETTING STARTED AN EXCERCISES

1.1 HELLOWORLD

4. Flutter and Kotlin are the two leading technologies used to build mobile applications. Kotlin has an easy learning curve because it is very similar to Java. In Flutter, developers must have to learn Dart programming to build an app.
5. In this laboratory students can use either kotlin / flutter.
6. Install Android studio, visual studio with kotlin / flutter on your machine.
7. Write a Hello-world program using android studio and kotlin / flutter source-code editor.

1. 2. FOOD ORDERING APPLICATION

Build a food ordering application with the following functionalities:

1. A 'Welcome page' which displays the logo and/or name of the app.
2. A 'Login Page' which asks for users' mobile number and password.
3. A 'Registration Page' which enables users to sign up for the app.
4. A 'Forgot Password Page' which enables users to reset their password.
5. A 'Navigation Drawer' with the app logo and user name on top and menu options to open the following pages:
   a. Home
   b. User Profile
   c. Favorite Restaurants
   d. Order History
   e. Frequently Asked Questions (FAQs)
   f. Log out
6. A 'My Profile' page (where the user's name, phone number, and address is displayed).
7. A 'Favorites' page (where the list of all favorite restaurants is displayed).
8. An 'Order History' page which lists the previously placed orders of the user.
9. An 'FAQ' page which lists some frequently asked questions. You can create any random questions which you feel could be relevant for a food delivery application. (Min. 5 questions)
10. A 'Logout' functionality which takes the user to the login page.
11. A 'Restaurant Details' page which displays the menu items of that particular restaurant, each item's price and the option to add an item to cart.
12. A 'Cart' page which lists the items added to cart and the total amount to be paid.

See also for authentic understanding, click the link **https://www.swiggy.com; https://www.zomato.com/deliver-food; https://www.ubereats.com; https://www.dineout.co.in and so on. Each student has to refer any one of the web sites stated above.**

2. MUSIC PLAYER APPLICATION

Build a music player application with the following functionalities:

1. A 'Splash screen' (gradient background and app logo in center)
2. A 'Navigation drawer' with app logo section at the top along with links to 'All Songs', 'Favorites', 'Settings' and 'About Us'.
3. An 'All songs' screen (where of list all the tracks fetched from offline storage are displayed and user can sort the tracks by name or recently added). This will the home screen of the app.
4. The app should be able to fetch and play .mp3 and .wav files.
5. A 'Favorites' screen (where list of all the favorite songs is displayed)
6. A 'Settings' screen (where the 'Shake to change song' feature can be enabled or disabled)

7. An ‘About us’ screen (where we will display information about the app developer and the app version)
8. A ‘Now playing’ screen with following features:
   a. Track title and track artist
   b. Play / Pause button
   c. Next button
   d. Previous button
   e. Shuffle button
   f. Loop button
   g. Seek bar
   h. Mark track as favorite or unfavorite it
   i. Third party visualizer in upper half background
   j. A ‘Back to list’ button in the header which should take the user to the screen he came from (kind of like back button behavior).
   k. Shake to change song
9. A ‘Now playing’ bar at the bottom with name of the track playing and play or pause feature. This would appear if the user has moved from ‘Now playing’ screen to ‘All songs’ screen or ‘Favorites’ screen without pausing the track.
10. Background play. The app will continue playing the track if the app gets closed (not killed) without the music being paused.
11. A notification saying "A track is playing in the background" only if the app gets closed (not killed) without the music being paused.

    Primary color scheme: #9b2a58, #00032a

See also for authentic understanding, click the link **https://open.spotify.com; https://gaana.com; https://www.hungama.com; https://wynk.in/music and so on. Each student has to refer any one of the web sites tated above.**

3. SMART HEALTH PREDICTION

Build Android Smart Health Prediction application which can be used by all patients or their family members who need help in emergency with the following functionalities:

This application comprises of 3 major modules with their sub-modules:

1. A 'User page' with menu options to open the following pages:
    a. A 'Patient Login page' which displays patient Login to the application using his ID and Password.
    b. A 'Patient Registration page' which asks if patient is a new user, he will enter his personal details and he will user Id and password through which he can login to the application.
    c. A 'My Details page' which patient can view his personal details.
    d. A 'Disease Prediction page' patient will specify the symptoms caused due to his illness. Application will ask certain question regarding his illness and application predict the disease based on the symptoms specified by the patient and application will also suggest doctors based on the disease.
    e. A 'Search Doctor page' which the patient can search for doctor by specifying name, address or type.
    f. A 'Feedback page' which the patient will give feedback this will be reported to the admin.
    g. A 'Notification page' which the user needs to enter his/her prescription and then the user need to select a time on which he/she wants to receive notification. The application will send the notification regarding the prescribed medicine to the user on their device.

2. A 'Doctor page' with menu options to open the following pages:
    a. A 'Doctor Login page' will access the application using his User ID and Password.
    b. A 'Patient Details page' can view patient's personal details.
    c. A 'Patient's Previous Details page' will get all information about patient's previous case history. That will help him to serve him better.

3. A 'Admin page' with menu options to open the following pages:
    a. A 'Admin Login page' can login to the application using his ID and Password.
    b. A 'Add Doctor page' can add new doctor details into the database.
    c. A 'Add Disease page' can add disease details along with symptoms and type.
    d. A 'View Doctor page' can view various Doctors along with their personal details.
    e. A 'View Disease page' can view various diseases details stored in database.
    f. A 'View Patient page' can view various patient details who had accessed the application.
    g. A 'View Feedback page' can view feedback provided by various users.

See also for authentic understanding, click the link **https://www.apollohospitals.com; https://nirogstreet.com; https://www.lybrate.com**; **https://www.portea.com and so on. Each student has to refer any one of the web sites stated above.**

4. HOSTEL MANAGEMENT APPLICATION

Develop the hostel management application which will help to manage the hostel. The hostel managers can keep track of the hostellers' in and out timings and their daily entries. This system is intended to help hostel admin by allowing them to save student records and information about their rooms. It helps the admin from the manual work from which it is very difficult to find the record of the students.

This application comprises of following functionalities:

1. A 'Admin page' with menu options to open the following pages:
    a. A 'Manage Rooms' page which the admin can choose to add general rooms or bedrooms to the system. If the admin chooses a bedroom, they can add beds. They can add, update or delete rooms.
    b. A 'Manage Students' page which the admin can view, add, update or delete students from the system. They can allocate rooms to the students. Also, they can view the attendance of a student.
    c. A 'View Attendance' page which the admin can filter by date AND room OR student-id. They can view all the student's attendance.

2. A 'Student page' with menu options to open the following pages:
    a. A 'Login' page which the student can log in to the system using a username and password.
    b. A 'Profile' page which t he student can add or update their profile details.
    c. A 'Change Password' which they can also change their password to the new one.

d. A 'Home' page the student can scan the QR Code and a log will be added. They can view their room allocation details.
e. A 'View Attendance' page the students are able to view only their attendance, they can also filter it by date.

3. A 'Guardian page' which the ward wishes to go out from campus for any purpose, the guardian will login through the registered login then the guardian will send the permission request to warden's android phone by specifying the purpose and also duration. After successful sent, the message is available in
the warden login for further action.
4. A 'Warden page' after receiving the guardian's message, warden has a choice with options approve and reject. Based on the purpose of out campus, Warden Selects approve or reject. If approves then the message with the ward details is sent to security check which indicates the request made by the guardian is permitted.

See also **for** authentic understanding, click the link **https://www.hostelsnap.com; http://www.ifnoss.com/edu/college-university-software/hostel-management-system.aspx; https://www.ezeetechnosys.com/absolute; https://www.resbird.com and so on. Each student has to refer any one of the web sites stated above.**

5. STAY SAFE WOMEN SECURITY APPLICATION

Build stay safe women security project is used to provide highly reliable security system for the safety of women. The proposed system is based upon advanced sensors and GPS. The basic aim of the system is to develop a low-cost solution for GPS based women tracking system (Women Safety System). The main objective of the system is to track the current location of the person which has an android enabled mobile by extracting the longitude and latitude of that target person.

This application comprises of following functionalities:

1. A 'Scream Alarm' page used perfect for the females as well as other users that need some kind of safety alarm in case, they found out that someone is following or stalking them. It also consists of two other types of scream alarm. It's an initial distraction which will buy some time and allow the user to escape from the trouble.

   a. Male voice scream
   b. Police siren.

   The user could select one of his/her choice from the "Settings" of the application, as keeping in mind the two other scream alarms are also added in this application as nowadays safety and security is everybody's concern.

2. A 'Fake Call Timer' page which the fake call timer allows the user to make fake calls in the time of need. It helps user to escape from an undesirable situation citing an important call from anyone who needs him/her urgently and rest depends upon user creativity. This feature also helps the user to escape from boring social events

In order to make a fake call the user have to select the "Fake Call" icon and after that user could write any name from which he/she wants a fake call. User could also set up the timer as per the requirement. The user could also set the default timer from the "Settings" icon of the application.

In a critical situation, the user just has to long term press the fake call button and automatically get a fake call as per the desired selected timer in the settings.

3. A 'Where Are You' which is used to find track friend. While first request is send by the sender. The sender will have to select the "Where Are You" icon and then a new dialog box of "Pick a Friend" will open up. The sender could select any friend and the request will be sent to the receiver. The receiver will accept that request from their end and a message will be sent to the receiver with the present location of the user.

4. A 'Track Me' which will track the user to view the exact dynamic location of the victim. First user have to send the Track Me request at the receivers end. The receiver will accept the request and then his/her name will appear on the friends you are tracking on the bottom of the application. The user could select that friend from there and then it will get automatically re-directed to the Google maps from where the user could view the exact location of the victim and also where's he/she heading to.

5. A 'Friends List' page which shows all the contact numbers of family and friends which are added by the user through contacts. This could be done by selecting the contact icon on the bottom right corner of the friends list.

6. A 'Settings' page which consists of the following features -:

   a. A 'Emergency Services' page allows the Stay Safe Application to send emergency notifications and SMS with the exact location to the emergency contacts.
   b. A 'Low Battery Alert' page alert feature allows the Stay Safe Application to send low battery alert and

SMS to the emergency contacts.

c. A 'Set Scream Sound' page which the user could select any scream sound as per the requirement.

d. A 'Fake Call Timer (On Long press)' page which the user could set the fake call default timer as per the requirement.

7. A 'Emergency Distress Signal (SOS)' which the distress signal will be generated by the user in case of an emergency. In order to generate the distress, signal the user have to shake up his/her phone, then a distress signal will appear at the user end with a default timer of 5 sec. In the end distress signal will be sent to the emergency contacts added by the user at the time of registration. The application sends SMS and user details as well as the exact location of the user through a push notification at the receiver end, before sending a distress signal the user first have to turn on the emergency services from the settings of the application.

When user launches the application in his/her Android phone, the very first screen which lands is the Login Screen. First the user have to register himself by entering the details as the respective name and contact number of the user.

See also for authentic understanding, click the link **https://womensafetywing.telangana.gov.in; https://nirbhayaapp.com; https://safetipin.com; https://www.smart24x7.com and so on. Each student has to refer any one of the web sites stated above.**

6. CONTROLLING ANTI RAGGING APPLICATION

Controlling anti ragging system facilitate college students to register a criticism in opposition to ragging immediately. Students would be required to log in, after which they would be capable to register their complaints. The grievance will then be dispatched to the worried authorities for well-timed action and the motion will be initiated immediately. Previous archives reveal that well timed action was once taken in every case, which in flip resulted in a fall in such cases. Now, with the launch of the app, the Ministry hopes to wipe out the exercise completely.

This application comprises of following functionalities:

a. A 'User Module' page which the person can register with the machine then he/she will get the get entry to in the app. After login done if the any ragging is happened then without delay update the details. Every other flexibility is additionally on hand right here to contact the emergency wide variety also provided.

b. A 'College Admin' page which can operate the action on the failed complaints; complaints can be considered via the admin and can take the perfect action on the complaint.

    a. A 'Add Complaint' page users can a make complaint against ragging. In this, those who are filling complaint as to fill some fields like Accused Name, Ragging Type, Mobile Number

    b. A 'View Complaint' page which can view the complaints made by students.

See also for authentic understanding, click the link http://www.amanmovement.org; **https://www.nmc.org.in/ActivitiWebClient/open/initiateAntiRaggingHome; https://www.antiragging.in; https://www.amanmovement.org/raggingmain.html and so on. Each student has to refer any one of the web sites stated above.**

7. EXTRACURRICULAR EVENT TRACKING APPLICATION

Build Extracurricular activities are sources that have long been a part of the educational system; students participate in these activities, which are not part of the standard curriculum or teaching techniques. Participation in all of these activities, or even just one of them has been linked to social and academic success. Students who engage in extracurricular activities gain from the numerous options available to them. Having better grades, higher standardized test scores and educational attainment, attending school more consistently, and having a greater self-concept were all advantages of participating in extracurricular activities.

Our Extracurricular Event Tracking System is developed to track students' hours organizing and participating in cultural events and college fests. This system keeps students updated about upcoming events and maintains track of the events they have attended and the hours they have spent organizing various cultural programs. Thereby, encouraging participation.

This application comprises of following functionalities:

1. A 'Admin' page which contains the following features:

    a. A 'Login' page which the admin can log in using their credentials.

    b. A Manage Course which they can add, update, view and delete courses. The courses will be BSC/MSC/Diploma/Engineering, etc. They can add total hours.

    c. A 'Manage Students' page which the admin can add, update, view and delete students. They can assign courses to the students.

    d. A 'Manage Events' page which the admin can add, update, view and delete events. They can add details about a particular event. They can choose courses. They can add the link to the Meet.

    e. A 'View Records' page which to view the records, the admin can choose by course. They can view the students' list and list of records.

2. A 'Student' page which contains the following features:
   a. A 'Login' page which the students can log in using their credentials.
   b. A 'Dashboard' page which the students can see the hours completed or the total hours. They can see up to 5 upcoming events ordered by date and time.
   c. A 'Calendar' page the students can see the Day/Week/Month on the Calendar.
   - Day: The system will show today's events on the date. The user can also select any previous date to check any events that happened on that day.
   - Week: By selecting any week, the system will show one week's dates & events.
   - Month: By selecting the month, the system will show the current month's dates and events.
   d. A 'Records' page which the system will show all the events attended.
   e. A 'Profile' page which the user can view their profile and change the password.
   f. A 'Forgot Password' page which if the user forgets their password, an OTP will be sent via Email or Phone number (Any 1). They would require to enter the correct OTP and reset the password.
   g. A 'Notification' page which the system will send a notification to the user before an event starts, and the dashboard checks the event unattended or hours not completed.

See also for authentic understanding, click the link
https://www.k12.com/parent-student-resources/extracurriculars-events,
https://www.tes.com/for-schools/clubs-and-events/features,
https://www.london.edu/masters-degrees/activities-clubs-and-groups,
https://prepory.com/blog/best-extracurricular-activities-for-the-engineering-applicant and so on.
Each student has to refer any one of the web sites stated above.

8. STUDENT MANAGEMENT SYSTEM

Build a student management system will have data on every student, and check the daily attendance of every student. The system comprises 3 major modules with their sub-modules as follows:

1. The 'Admin' page menu options to open the following pages
   a. A 'Login' page the admin can log in to the system using a username and password.
   b. A 'Manage Students' page the admin can manage student details and they can add, update and delete details. Also, add a fingerprint while adding the student's details.
   c. A 'Manage Teacher' page admin will also add teacher's details and add, update and delete details. Teachers will be assigned subjects.
   d. A 'Manage Grades' page can manage students' grades and add, update or delete grades.
   e. A 'Manage Subjects' page can also manage subjects by adding, updating and deleting subjects.
   f. A 'Manage Appointments' page applies to meet parent and waits for a reply.
   g. A 'Appointments' page parents applied to meet the teacher/admin. The admin will respond to the parent's appointment request.
   h. A 'Manage Leaves' page able to manage pending and past student leave. They can view a list of all the applied student's leaves. The admin can view all the pending leaves applied by parents. The admin will approve or reject the leave application. Leave Note: They will be able to send a note to the parent related to leaves.
   i. A 'Medical Certificate' page will be able to view the medical certificate submitted by parents. They can send a note with an acknowledgement in response.
   j. A 'View Complaint' page can view pending and replied complaints. They will be able to respond to the complaints.
   k. A 'Warning Letters' page contains the admin can add and view a list of all students with attendance below 80% / total course work below 60%. They can send a warning letter to their parents. Admin will be able to view all warning letters and replies by parent.
   l. A 'Attendance' page contains Entry and Exit Lecture-wise Attendance. See the attendance list of all Students grade-wise against 2 dates.
   m. A 'Reports' page contains can view reports of all the student's attendance lists, grade-wise. They can view students' attendance reports for 2,4,6-months with lecture, entry and exit details. They can also view the academic reports of all the students.
   n. A 'Entry/Exit' page can add entry and exit of the student biometric.

2. A 'PARENTS' page contains following features
   a. A 'Login' page can log in to the system using login credentials.
   b. A 'Profile' page can add or update details to their profile.
   c. A 'Change Password' can also change their old password to the new one.
   d. A 'Forget Password' used if they forget their login password, they can receive a link to change the password

on their registered email-id.

e. A ‘Appointments’ page contains the parent can apply to meet the admin/teacher and waits for their response. The parent will also receive the admin’s request to meet them and they can respond to the request.
f. A ‘Leaves’ page contains Parents can view all the leaves including pending leaves. They can view the list of all their applied leaves. Parents can view leaves that have not been replied to by an admin.They can also cancel their applied leaves.
g. A ‘Medical Certificate’ page contains Add: Parents can upload the medical reports of their child while applying for leave. They can view all the previously uploaded lists and replies from the admin.
h. A ‘Complaint’ page contains Parents can add a new complaint. They can view a list of all the complaints and replies from the admin.
i. A ‘Warnings’ page list of the warning letters sent by admin and parents can respond to them.
j. A ‘Attendance’ page can view their child’s attendance, lecture-wise. Also, they can view the attendance list of the child against 2 dates.
k. A ‘Reports’ page contains following features: Academic attendance reports can be viewed by the admin. Parents can view attendance reports of 2/4/6 Months with lecture, entry and exit details. They can also view their child’s academic report.
l. A ‘Notifications’ page will receive notifications such as when appointment status changes or admin applied, leaves approved/rejected, medical certificate reply, complaint reply, warning letter, student entry/exit, academic marks uploaded by teacher.

3. A ‘TEACHER’ page contains following features
   a. A ‘Login’ page can log in to the system using a username and password.
   b. A ‘Profile’ page can add or update details to their profile.
   c. A ‘Change Password’ can also change their passwords.
   d. A ‘Forget Password’ if they forget their login password, they can receive a link to change the password on their registered email-id.
   e. A ‘My Subjects’ page contains following features
      Assigned Subjects: A list of all assigned subjects can be viewed grade-wise.
      Students List: The teacher can view the list of the students in that Grade.
   m. A ‘Attendance’ page contains they can select grade and subject to view attendance. Biometric Attendance: They can take the biometric attendance of students. See the Attendance list of all Students grade-wise against 2 dates, lecture-wise.
   n. A ‘Academics’ page contains teachers can choose the grade, subject and student for their academic reports. They can view, add and update students’ academic details. Teachers can add marks on tests, assignments and final exams. They can update existing marks on tests, assignments and final exams.
   o. A ‘Complaint’ page contains teachers can add new student complaint. They can view list of all the complaints and replies from admin.

See also for authentic understanding, click the link
https://www.iitms.co.in/products/student-information-system-sis; https://samvidha.iare.ac.in; https://themeforest.net/item/clever-course-learning-management-system-theme/8645312; https://markerspro.in and so on. Each student has to refer any one of the web sites stated above.

9. PHARM EASY APPLICATION

Build a pharm easy application for ordering medicines and tests. This application comprises of following functionalities

1. A “Home Page” which consists of a list medicines & health care product.
2. A “Lab Tests page” contains different medical tests.
3. A “Health Care Products page” contains vitamins and necessary to our body.
4. A “Offers Page” offers on different products and coupons.
5. A “Account Page” contains details of individual customer.
6. A “Products Page” contains a list of products.
7. A “Detailed View of Product” detailed view of selected product.
8. A “Cart page” list of items we selected to buy.
9. A “Delivery Details Page” consists of customer address and contact details.
10. A “Payment Page” how to make payments (cod or online payments).
11. A “Success page” show transaction success.
12. A “Bottom APP Bar” consists of logos which opens the corresponding pages(fragments).
    a. Home page
    b. Test Page

c. Health Care Page
d. Offers Page
e. Account Page

See also for authentic understanding, click the link **https://pharmeasy.in; https://www.netmeds.com; https://www.medlife.com; https://www.practo.com and so on. Each student has to refer any one of the web sites stated above.**

10. NEWS APPLICATION

Build a news application is to connect news articles from all around the world and deliver it to user as fast as possible in best visualize way. This application comprises of following functionalities:

1. A 'User Interface' page should be able to select from different categories, countries and newspaper. Short News as list view with header, little description and image before showing full article can be helpful to user to determine what type of news they are looking for. View Holder can be used for this list view for better and fast experience.
2. A 'Admin Panel' page controls the User and Writers logins from database. Writers can add news, update and delete from its database as per required. Main Admin can add Users, Writers, and News. He can also approve, update and delete it. Using this approach, we can create network in local areas connect by writers and local admins which will provide news at local level and we can also implement location feature which will update local news of different location or city.
3. A 'Global Support' page contains different type of newspaper will be available from all around the world in different languages with this user will be able to get news from all around the world.
4. A 'Short News' page News will be displayed in short format with title, image and little description in list view. It will help user to access required news faster.
5. A 'Search Option' page which user will be able to search from not only one source but many different sources available within API.
6. A 'Favorites / Offline Reading' page which News can be added as favorites which will automatically will be saved for offline reading.
7. A 'Sharing' page which user will be able to share news easily on social media.

See also for authentic understanding, click the link **https://indianexpress.com; https://www.bbc.com/news; https://www.ndtv.com; https://www.hindustantimes.com and so on. Each student has to refer any one of the web sites stated above.**

11. AIR TRANSIT TRIP PLANNER APP

Build the air transit trip planner which This passenger on transit will spend their time at the airport while waiting for the next flight without any planning to visit tourism places nearby due to short period of time and much information needed to plan a short trip. This is such a waste because they actually can boost countries' economy in the tourism sector if they spend their time to visit some tourism places nearby the airport. Because of that, Air Transit Trip Planner aims to help these passengers on transit by planning a short trip to the nearby tourism places within the transition hour they have.
Air Transit Trip Planner also using Google Map API as a map to guide user and Google Places API as a data center to grab all the tourism places and its details. The output of this calculation is a suggestion of short trip planner for the user (passenger on transit). The trip planner will list down the tourism places user can visit within their flight transition hour, the distance to go there and also time taken to go there. Indirectly, this Air Transit Trip Planner application also helps to boost economy in the tourism sector of a country.
This application comprises of following functionalities:

1. A 'Main Page' which the user will see when they launch the application. In the background when this interface prompts, the system actually will automatically detect user current location by using GPS or Google Places. If no internet access or GPS is turn off, the system will prompt a pop up asking the user to turn on GPS or connect to internet. But if everything is fine, user just need to key in their transit hour and departure time then click enter.
2. A 'Tourism places' page will calculate how many tourisms places the user can visit within the transit hour they have by using the algorithm.
3. A 'place details" page will direct to new user interface with scrollable list view.
4. A 'pop up' page when user click each mark on map. When the user clicks the mark, the pop up will show the address of each location.
5. A 'GPS Navigation' page is using satellite and it is implementing context awareness concept where it will direct user to their destination. This interface also has voice navigation which will help the user to hear the navigation while driving rather than looking at the map continuously.

See also for authentic understanding, click the link **https://travelplanner.co.in; https://www.expedia.com; https://www.orbitz.com; https://www.kayak.com and so on. Each student has to refer any one of the web sites stated above.**

12. STUDENT-FACULTY DOCUMENT SHARING SYSTEM

Develop Student-Faculty Document Sharing System. This application is an online portal between students and faculty. This innovative system allows college faculty to share important data as well as notifications with engineering students. It consists of a faculty login along with student login. Since college faculty operate through pc and document uploading is simpler through a pc, the faculty login is to be performed through a computer. Faculty may upload documents of subject syllabus, timetable document, notifications, notes etc through their provided login. The documents are uploaded by faculty to different corresponding departments. We propose to build this system on an online server that allows faculty to upload data and students may view search and download required documents through their android device. Here students only see and download data of their particular semester. Faculty may access and upload/edit documents to any semester or add any notice as desired.

This application comprises of following functionalities:

1. A 'User login' page allows only the registered users to login in order to use this location tracking application.

2. A 'Document details' page stores documents in word and pdf format and also allows students to view particular data.
3. A 'Server management' page which displays the server smartly handles data and allows students with an android device to access data as well as faculty to upload document details.
4. A 'Faculty Upload' page allows faculty to upload documents to server.
5. A 'Student Download' page allows student to download only allowed semester data through his android phone using an active internet connection

See also for authentic understanding, click the link **https://akanksha.iare.ac.in; https://moodle.org; https://classroom.google.com;**
https://nevonprojects.com/student-faculty-document-sharing-android-project and so on.
Each student has to refer any one of the web sites stated above.

13. ONLINE RECRUITMENT SYSTEM

Build online recruitment system with the following functionalities:

1. A 'Administrator' page stores the information of the user's login and the password data. This provides security to the system and keeps the record of which user entered in the system at what instance of time. **It has the following attributes:**
   a. Username: It stores the name of the admin which acts as the unique name given to the manager of the firm.
   b. Password: This attribute holds the secured keyword given to every manager of the educational institutions who need access to the system
   c. Login-Time: The login time of the admin will be recorded in this field which helps in tracking the admin performance.
   d. Logout-Time: It stores the logout time of the admin from the system
2. A 'Job Seeker' page stores the information of the candidate who registers to the system for participation in the recruitment process, the Job Seeker entity is created. It will have all the required data about the applicant. I**t has following attributes**
   a. Can-Id: Candidate Id is a unique number given to each candidate who registers in the system
   b. Can-Name: Candidate Name is the information required for details of the applicant who registers in the system.
   c. Resume/CV: In this attribute job, seekers can store their resumes which can be reviewed by the organization if they want to further check the candidate
   d. Projects: Every candidate has many projects which they have created or worked on, which they want to share with them, recruiters.
   e. Personal-Info: This attribute holds all the information the recruiters needed from the candidate apart from its technical skills and eligibility.
   f. Contact: This stores the contact information of the candidate like phone number and email id
3. A 'Skill' page is an essential part of the system as it works on both from recruiter and job seeker.
   It has the following attributes
   a. Skill-Id: As every set of skills is unique so every set is given a unique number that acts as its id in the system.
   b. Technical: It will store the technical skill by the applicant.

   c. Academic-details: It will store the detailed information of the student in an academic field like high school marks, senior secondary marks. It will also have the graduation percentage is necessary for the skill.
   d. Cocurricular-activities: Every skill does not only require the technical talent of the student it also needs other social skills.
4. A 'Requisition' page every employer will make the requisition of the application for the given post. From the application received from the number of the job seekers, the company will check their eligibility and decide whether he/she is compatible for the post or not. Then the shortlisted student will go for further rounds of the recruitment process as per the employer. **It has following attributes**
   a. Req-Id: Every requisition present in the system given by each employer has unique numbers assigned to it.
   b. Emp-Id: This attribute will hold the information of the employer who is giving this requisition.
   c. Post: Every requisition given by the employer belongs to a specific post
   d. Skill-Id: It is a reference to a skilled entity that implies the required skill set for the specific post.
   e. Package: This attribute stores the information regarding the pay scale or salary
   f. Criteria: In the criteria attribute, data regarding the process of recruitment is stored.
5. A 'Employer' page will store the detailed information of the employer present or registered in the system. Thus, making it one of the essential entities in the system. Before applying for any post, the applicant also wants to know about the company or the employer who has given the requisition. **It has the following attributes**
   a. Emp-Id: It is the unique number given to the employer to differentiate from others.
   b. Emp-Name: It will hold the name of the employer which is registered in the system.
   c. Information: This entity will hold the information of the employer like when it is founded, what kind of projects they are working on
   d. Key-Persons: It holds the data of the important persons of the company and the point of the contact for the interested candidates.
6. A 'Interview' page has information about the interviews that will happen for the specific requisition. As the interview is an important part of any recruitment process as in this the employer and candidates come face to face and they can judge each other. **It has following attributes**
   a. Int-id: It is the unique number given to each interview process that happens in the system
   b. Int-type-id: As there are many kinds of interviews that can happen during the recruitment process.
   c. Req-Id: It will hold the reference of the requisition for which a specific employer is conducting this interview.
   d. Remarks: This has the remarks of the interviews which is given by the employers.
   e. Emp-Id: It holds the data of the employer who is conducting this interview and managing the whole process.
7. A 'Interview-Type' as there is various type of interview process which happens in the recruitment like technical for checking the technical knowledge of the candidate or the HR interview to test the social and interpersonal skill of the applicant. **It has following attributes**
   a. Int-type-id: It is the unique number given to each type of interview
   b. Int-type-Name: It holds the name of the type of interview.
8. A 'Offer-Letter' when candidates apply for the job and it takes every criterion provided for a specific post. If it qualifies for the post the company will provide him/her with the offer letter. It is a document that makes the offer to the candidate for joining the employer's firm. It has detailed information regarding the package and the work environment of the company.

It has following attributes

   a. Of-id: Each offer letter is given a unique and distinct number
   b. Emp-Id: It is the number given to the employer as it gives the information about which company has given this letter.
9. A 'Experience' to show the talent each applicant writes the experience they have gain in the previous years. It helps the employer to see which applicant has the relevant knowledge required for the post. So, they can hire a person who has more potential than others. **It has following attributes**
   a. Exp-Details: It holds the information candidate has learned during the work.
   b. Exp-Organization: The name of the organization in which the applicant has gained the experience.
10. A 'Feedback' page is holds the feedback of the job seekers about their recruitment process for an employer or a requisition. It has the following attributes:
    a. Feed-Id: It is a unique number given to each feedback. It makes the remark distinct and identifiable for the employers to take note of.
    b. Emp-id: It is a reference to the employer for which this feedback is given
    c. Feed-details: It holds the remark given by the applicant

See also for authentic understanding, click the link **https://www.tracker-rms.com;**

https://www.bullhorn.com/glossary/online-recruitment-system; https://www.manatal.com; https://www.whatishumanresource.com and so on. Each student has to refer any one of the web sites stated above.

14. STUDENT COUNSELING MANAGEMENT SYSTEM

Build Student Counseling Management System with the following functionalities

1. A 'Student' page will store the record of the student registered for the counseling. It is essential for the system as a separate unit which has all the information regarding the student and their personal and professional data. So, it will have the 10th and 12th percentage, rank in the entrance exam will also be asked from student to give the preferences according to it. It has the following attributes:
   a. Name: This will store the name of the student as it needed while saving the required
   b. DOB: The date of birth of the student as mention in the high school certificate and will act as the candidate key in the database model
   c. Student-id: It is a unique alphanumeric number assigned to every student who registers for the counseling
   d. Password: It is an alphanumeric field that has the constraint of having a minimum of 8 digits and at least one alphabet with numbers
   e. 10th Percentage: This is used to get the student's percentage in high school which some universities check before giving admission to the student.
   f. 12th Percentage: It is an academic field that requires the student to give information about the 12th standard.
   g. Rank-Entrance-Exam: This attribute gets the entrance exam rank from the student which they have got according to their performance in the exam
2. A 'Choices' page is needed to get the preferences from the student based on their rank in the entrance exam. The number of choices given to the student can be multiple based on the institution that requires the seats available for the courses. These preferences will go to the courses entity which then checks which selection is eligible for the student, this evaluation will be based on the number of seats available and the rank of the student. This is also will be referenced in the allotment of the seats where the allotted seats are recorded. It has the following attribute:
   a. Rank-Entrance-Exam: This attribute gets the entrance exam rank from the student which they have got according to their performance in the exam.
   b. 1st Preference: This will be a simple attribute that holds the first preference of the student which he/she will select from the pool of choices.
   c. 2nd Preference: It will hold the second preference of the student as it is not always possible to get the first selected choice as it may have got filled or the student with a higher rank got that seat.
3. A 'Courses' page will have all the courses provided by the institutes. All the courses will have institute code will differentiate if the same courses are provided by the different institution. Every course has the define the seats available and the allotted seats along with the total seats available which will provide transparency in the system. It has the following attribute:
   a. Course-code: A unique number given to every course not depends on which university to it belongs, it helps in maintaining the record unique and distinct so act as the primary key for the entity.
   b. Institute-code: Every institute that has register itself for the system of counseling has given a unique code that differentiates it from other institutes which have the same courses.
   c. Total-seats : It will have the total seats available for that course provided by the Institute.
   d. No-of-seats: It is a composite attribute which has the further information about a number of seats allotted and the vacant seat left for the course.
4. A 'Course-Name' page will contain the name of the course and the unique id of that course. As the many institutes provide the same course it would be a difficult job to assign a different code to the same course. So, this field will be normalized. It has the following attribute:
   a. Course-code: A unique number is given to every unique course.
   b. Course-name: The name of the course will be stored in this attribute.
5. A 'Allotment of Seat' page will store the information of the student after the seat is confirmed for allotment. It has the following attribute:
   a. Student-id: Student unique id number.
   b. Rank-Entrance-exam: Rank gave to the student as per his/her performance in the entrance exam
   c. Preference-no.: The preference which got allotted, as per the choices are given to him.
   d. Course-code: The course code which got selected for.
   e. Institute-code: The institute allotted the seat.
6. A 'Institutes' page will store the information regarding the institutes which are participating in the counseling. Each one of them has to publish their courses for which they want students with the number of the total seats. It has the following attributes:

a. Institute-code: Every institute that has register itself for the system of counseling has given a unique code that differentiates it from other institutes which have the same courses.
b. Institute-name: It will hold the name of the institute as mention by them in the registration process.
c. City: It depicts the city to which the institute belongs to as it helps as a candidate key.
d. Password : It is an alphanumeric field that has the constraint of having a minimum of 8 digits and at least one alphabet with numbers.

7. A ‘Admitted-Student’ page maintains their own record of which student has admitted to their institute and the courses what courses they have opted for. It has the following attributes:
   a. Student-id: Student unique id number.
   b. Course-code : The course code which got selected for.
   c. Institute-code: The Institute for which the seat is allotted. Institute is related to this entity as every university will have a database of how many students have been admitted to its courses.

See also for authentic understanding, click the link **https://www.academiaerp.com; https://collegedunia.com/exams/cuet/counselling; https://nchmcounselling.nic.in; https://www.cdac.in/index.aspx?id=edu_acts_Login_cv11 and so on. Each student has to refer any one of the web sites stated above.**

15. DATA MART MANAGEMENT SYSTEM

Build a Data Mart Management system with the following functionalities

1. A ‘Master Maintenance’ page maintains the following master details for various purposes. It has the following attributes
   a. Suppliers Details
   b. Sub location In-charges Details
   c. Retailers Details
   d. Products Details
2. A ‘Shipment’ page provides you a number of functions to maintain the flow of goods in warehouse. It has the following attributes
   a. Transfer inventory
   b. Put inventory on hold
   c. View inventory balances
   d. View inventory transactions
3. A ‘Billing’ page warehouse billing process allows the retailer to generate new bills and also allows viewing of existing bills.
4. A ‘Reports’ page reports process allows the user to produce the reports necessary for day to day warehouse operations. The standard reports are
   a. Inventory Reports
   b. Purchase Order Reports
   c. Administrative Reports

See also for authentic understanding, click the link **http://www.datamartsys.com; https://netresultsgroup.com; https://www.indiamart.com; https://uniondatamart.com and so on. Each student has to refer any one of the web sites stated above.**

16. RESTAURANT RESERVATION AND TABLE MANAGEMENT SYSTEM

Build a Restaurant Reservation and Table Management system is a Web application designed to help you (and your coworkers) to select the restaurant you are going to eat in. You can manage users, restaurants, menus, prices, give notations to each lunch, etc.

1. A ‘Reservation Management’ page functionalities are
   a. Easily enter or modify reservations while viewing guest histories.
   b. Capture phone numbers, email and mailing addresses.
   c. Allow management blocking and VIP pre-assignments.
   d. Reduce no-shows with enhanced customer tracking.
   e. Take reservations from your website or Open Table 24 hours.
2. A ‘Table Management’ page functionalities are
   a. Maximize seat utilization with walk-in and waitlist functionality.
   b. Instantly track covers for more efficient kitchen and server management. Increase table turns by tracking party status. Store multiple reservation sheets for holidays and special events.
   c. Hold and combine tables for large parties.

d. Record and view shift notes for each day

3. A ‘Guest Management’ page functionalities are
   a. Identify regulars and VIPs
   b. Track customer preferences to meet and anticipate special requests
   c. View customer reservation histories at-a-glance
   d. Track special occasions such as guest birthdays and anniversaries Marketing Management
   e. Conduct powerful email marketing campaigns to increase repeat business.
   f. Print mailing labels to reach select target audiences.
   g. Track and reward concierge business.
4. A ‘Increase control’ page functionalities are
   a. Manage reservations from the back-office or any other location. Simultaneously control multiple restaurants from key centralized locations.
   b. Share guest data across sister restaurants.

See also for authentic understanding, click the https://www.elluminatiinc.com/restaurant-reservation-system; https://restaurant.opentable.com; https://www.tornosubitodubai.com; https://indiarestaurant.co.in **and so on. Each student has to refer any one of the web sites stated above.**

17. SECURE STOCK EXCHANGE SYSTEM

Build a secure stock exchange system that is designed for the sale and purchase of securities of corporations and municipalities. A stock exchange sells and buys stocks, shares, and other such securities. In addition, the stock exchange sometimes buys and sells certificates representing commodities of trade.

The system “Secure Stock Exchange System using Web Services” consists of 3 modules

1. A ‘Stock Markets and Investments’ page functionalities are
   a. Stock Exchange Listing
   b. Stock Options & Analysis
   c. Stock Market Crash
   d. Selling Stock Certificates
   e. Stock Market Forecasts
2. A ‘Stock Options’ page contains different types of Stocks
   a. Stock Option Valuation
   b. Restricted Stock Options
3. A ‘Related Information’ page functionalities are
   a. Day Trading Stocks
   b. Stock Quotes & Stock Ticker
   c. Stock Charts
   d. Share Portfolio Management

See also for authentic understanding, click the https://stocksandsecurities.adityabirlacapital.com; https://www.nseindia.com; https://www.jpx.co.jp/english; https://www.mstock.com **and so on. Each student has to refer any one of the web sites stated above.**

18. COUNTRY CARGO AND EXPRESS COURIERS SYSTEM

Build a project country cargo and express couriers system to deal with the queries of manger. How much more time it will take to reach the place, and whether received in original state. Queries among the officers regarding handling etc. User satisfaction guest book where user should put some notes about service and other suggestions. User complaint registers to help us get better feedback for our failures as a hospitable interface. Providing an interactive interface for the user query for online status of the packets booked and delivered.

Modules:

1. A ‘LOGIN MASTER’ page is the specific module, which only has to deal with the updating of the database. Two types of user can login administrator and the employee. It checks for a valid candidate when the user enters his user id password and link to the correct page and link to the registration page. It is divided into following functions.
   a. USER ID CHECKER: As user id rules the system for updating (control panel). So his uniqueness and type of his user id know a person whether he is master or one of the employee. As it is to provide the viable candidate system, so the user id is being validated with password in different cases to validate the genuinely of the candidate.
   b. THE LOGIN STATUS MASTER: This module keeps the status of who and when logged in and for which purpose and for how much time.

2. A ‘REGISTRATION MASTER’ page as this module is only accessible through administrator password. This modules deal with the different state of registration as-
   a. Registration form display
   b. Client’s does validation being handles by validation master
   c. Unique user id checker (checks that the user id being entered by the candidate is unique or no)
   d. Auto user id generator (user id field by taking the email id of the user if it is unique or suggest by combining it will some number).

3. A ‘COURIER STATUS NOTIFICATION MASTER’ module handles the query of customer and displays the result according to that customer is asked to enter the booked id in the specified input area. On the basis of input the detail regarding where material exactly is shown.
   This module contains the following sub modules-
   a. QUERY HANDLER: This module handles to and from the courier and cargo differentiation master searches the data in the database, which through different conversion through numbers of tables shows the result regarding where exactly the cargo is?
   The booking id is first searched into the courier booked table. Each booking id has a unique number associated with the different offices and also with the courier.
   On the basis of that booking id, matched with the lot id, and then with the medium and after that medium halted at which station. These statuses are shown.

4. A ‘DELIVERY STATUS NOTIFICATION’ page module deals with the delivery status whether the following courier has been delivered to the destination or not, how much time it will take to reach the destination. It has following sub modules-
   a. DISTANCE CALCULATOR: It takes the data from the route table that specifies distance between the current city (the city through which tile courier is passing and the destination).
   b. TIME CALCULATOR: An average time is calculated on the basis of the average time taken to reach the two adjoining city subsequently to the destination.
   c. DELEVERED STATUS: It checks the current status if the current status of the courier gives the destination address and after taking the received detail. It display whether the courier is on its way or delivered successfully.

5. A ‘BOOKING DETAILS FOR CARGO’ page for cargo booked, source office, destination office, container, truck detail, date dispatched etc regarding booking of new courier with a unique id.
   It includes the following sub modules-
   a. CARGO STATUS : This specially deals with the updating at intermediate office about the truck number that passed through office.
   b. LOADING UNLOADING DETAILE :This deals with the loading and unloading cargo from one truck to another truck or one container to another container and other entries to keep the site working.

6. A ‘CUSTOMER QUERY HANDLER MASTER’ module specially deals with the handling of query of the customer. This module has following sub modules –
   a. CUSTOMER QUERY HANDLING : This module generates a input form through which customer can directly interact with tile intermediate office where there courier & cargo is! Through different function and tables.
   b. CUSTOMER COMPLAINT REGISTER : Customer can directly enter the complaint regarding end office, and the response is sending through the head office. This complaint is only viewable through administrator account.

7. A ‘CARGO STATUS NOTIFICATION MARKER’ page modules handles the query of customer and display the result according to that customer is asked to enter the booked id in the specified input area. On the basis of input the detail regarding where there material exactly is shown.
   This modules contains the following sub modules-
   a. QUERY HANDLER : This modules handles the query of the customer searches the data in the database, through number of tables shows the result regarding where exactly the cargo is? The booking id is first searched in to the cargo booked table. Each booking id has unique number associated with the different offices and also with the cargo. On the basis that booking id, matched with the container, and then with the truck, and after that true halted at which station. The status is shown.

8. A ‘DELIVERY STATUS NOTIFICATION SYSTEM FOR CARGO’ system deals with the delivery status whether the following cargo has been delivered to the destination or not or how much time it will

take to reach the destination. It has following sub modules-

a. DISTANCE CALCULATOR: It takes the data from the route table that specifies distance between the places then calculates the distance between the current cargo (the city through which the cargo is passing and the destination).
b. TIME CALCULATOR: An average time is calculated on the basis of the average time taken to reach the two adjoining city subsequently to the destination average truck halt time and a time in day returned.
c. DELEVERED STATUS: It checks the current status if the current status of the cargo gives the destination address and after taking the received detail. it display whether the cargo is delivered successfully or not.

See also for authentic understanding, click the https://worldwideexpresscouriers.com; https://www.vxpress.in; https://www.tciexpress.in; https://www.bluedart.com **and so on.**
Each student has to refer any one of the web sites stated above.

V. TEXT BOOKS:

1. Reto Meier, *Professional Android 4 Application Development*, Wile Publication, 1st Edition, 2012.

VI. REFERENCE BOOKS:

1. Bill Phillips and Chris Stewart, Kristin Marsicano *Android Programming*, The Big Nerd Ranch Guide, O'Reilly, 3rd Edition, 2017.
2. Dawn Griffiths, David Griffiths, *Head First Android Development: A Learner's Guide to Building Android Apps with Kotlin, Third Edition*, O'Reilly, 3rd Edition, 2021.
3. Antonio Leiva, *Kotlin for Android Developers: Learn Kotlin while developing an Android App*, CreateSpace Independent Publishing, 1st Edition, 2016.

VII. WEB REFERENCES:

1. https://www.javatpoint.com/android-tutorial
2. https://www.tutorialspoint.com/android/index.htm
3. https://docs.flutter.dev/
4. https://developer.android.com/courses/android-basics-compose/course
5. https://developer.android.com/guide

VIII. MATERIALS ONLINE

1. Course Template
2. Lab Manual

COURSE CONTENT

GENDER SENSITIZATION								
II Semester: AE / ME / CE / ECE / EEE / CSE / CSE (CS) / CSE(DS) / CSE (AI & ML) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSD10	Mandatory	L	T	P	C	CIA	SEE	Total
		-	-	-	-	-	-	-
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: Nil**			**Total Classes: Nil**			
Prerequisite:								

I. COURSE OVERVIEW:
The course aims at raising awareness of gender equality among students from sociological, cultural, psychological, legal and economical perspectives, thereby empowering them to communicate better in a cross-cultural work ambience. At the end of this course the students expose to better egalitarian interactions between men and women and to enable them see diversity and inclusiveness as assets in a globalized scenario.

II. COURSES OBJECTIVES:
The students will try to learn

I. The basic gender concepts and their application to the long- term implementation of programming and initiatives.
II. The gender roles, expectations and issues and their impact on as the community's day-to-day life.
III. The more egalitarian interactions between men and women.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO 1	Develop a better understanding of important issues related to gender in contemporary India.
CO 2	Sensitize to the basic dimensions of the biological, sociological, psychological and legal aspects of gender.
CO 3	Acquire insight into the gendered division of labor and its relation to politics and economics.
CO 4	Attain a finer grasp of how gender discrimination works in our society and how to counter it.
CO 5	Men, women and professionals will be better equipped to work and live together as equals.
CO 6	Know the new laws that provide protection and relief to women.

IV. SYLLABUS:

MODULE–I: UNDERSTANDING GENDER

Introduction: definition of gender, basic gender concepts and terminology, exploring attitudes towards gender, construction of gender. Socialization: making women, making men, preparing for womanhood. growing up male, first lessons in caste.

MODULE – II: GENDER ROLES AND RELATIONS

Two or many; struggles with discrimination; Gender roles and relations: types of gender roles, gender roles and relationships matrix, missing women, sex selection and its consequences, declining sex ratio, demographic consequences; Gender Spectrum: Beyond the Binary.

MODULE-III: GENDER AND LABOUR

Division and valuation of labour; Housework: the invisible labor, my mother doesn't work. Share the load work: its politics and economics, fact and fiction. unrecognized and unaccounted work; gender development issues gender, governance and sustainable development; gender and human rights; gender and mainstreaming.

MODULE–IV: GENDER AND BASED VIOLENCE

The concept of violence; types of gender; based violence, gender, based violence from a human Rights perspective, Sexual harassment: say no sexual harassment, not eve-teasing-, coping with everyday harassment, further reading, chupulu. Domestic violence: speaking out is home a safe place, when women unite (Film). rebuilding lives, thinking about sexual violence blaming the victim, i fought for my Life.

MODULE–V: GENDER AND CULTURE

Gender and film; gender and electronic media; gender and advertisement; gender and popular, literature, Gender development issues, gender issues, gender sensitive language, gender and popular literature. Just relationships: being together as equals; Mary kom and onler, love and acid just do not mix, love letters, mothers and fathers, Rosa Parks, the brave heart.

V. TEXT BOOKS:

1. A. Suneetha, Uma Bhrugubanda, DuggiralaVasanta, Rama Melkote, Vasudha Nagaraj, Asma Rasheed, Gogu Shyamala, Deepa Sreenivas and Susie Tharu The Textbook, *Towards a World of Equal: A Bilingual Textbook on X Gender*, published by Telugu Academy, Telangana Government, 1st Edition, 2015.

VI. REFERENCE BOOKS:

1. Kadambari V, *Gender Studies: A Primer. Rajiv Gandhi National Institute of Youth Development, Sriperumbudur.* 1st Edition, 2009.
2. C. Rajya Lakshmi Kalyani, D.S. Vittal, A. Kanaka Lakshmi, P. Chandrakala, B. Lavanya., *Gender Sensitization*, Himalaya Publishing House. 1st Edition, 2017.

VII.ELECTRONICSRESOURCES:

1. https://en.unesco.org/women-make-the-news-2017/resources
2. http://ncw.nic.in/sites/default/files/Booklet-%20Gender%20Sensitization_0.pdf

COURSE CONTENT

PROBABILITY AND STATISTICS								
III Semester: AE / ME / CE / CSE / CSE (AI&ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours/Week**			**Credits**	**Maximum Marks**		
AHSD11	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	1	-	4	40	60	100
Contact Classes: 48	**Tutorial Classes: 16**	**Practical Classes: Nil**				**Total Classes: 64**		
Prerequisite:								

I. COURSE OVERVIEW:

Probability theory is the branch of mathematics that deals with modelling uncertainty. The course includes: Baye's theorem, random variables, probability distributions, hypothesis testing, confidence interval and linear regression. The use of probability models and statistical methods is for analyzing data, designing, manufacturing a product and the observed class frequencies for engineering and sciences.

II. COURSE OBJECTIVES:

The students will try to learn:

I	The theory of random variables, basic random variate distributions and their applications.
II	The methods and techniques for quantifying the degree of closeness among two or more variables and the concept of linear regression analysis.
III	The estimation statistics and hypothesis testing which play a vital role in the assessment of the quality of the materials, products and ensuring the standards of the engineering process.
IV	The statistical tools which are essential for translating an engineering problem into probability model.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Explain the probability elementary theorems on probability conditional, multiplication, Baye's theorem under randomized probabilistic conditions.
CO 2	Apply the role of random variables and types of random variables, expected values of the discrete and continuous random variables under randomized probabilistic conditions
CO 3	Apply the parameters of random variable Probability distributions such as Binomial, Poisson by using their probability functions,
CO 4	Interpret the parameters of random variate Probability distributions such as Binomial, Poisson and Normal distribution by using their probability functions, expectation and variance
CO 5	Make Use of estimation statistics in computing confidence intervals by Correlation Analysis, Regression analysis
CO 6	Identify the role of statistical hypotheses, types of errors, confidence intervals, the tests of hypotheses for large and small sample, in making decisions over statistical claims in hypothesis testing.

IV. COURSE CONTENT:

MODULE-I: PROBABILITY (10)
Probability, axiomatic approach, elementary theorems on probability, conditional probability, multiplication theorem, Bayes theorem (without proof).

MODULE-II: RANDOM VARIABLES (09)
Random variables: Discrete and continuous random variables, probability distribution, probability mass function and probability density function.

MODULE-III: PROBABILITY DISTRIBUTION (10)
Binomial distribution: Mean and variance of Binomial distribution, Poisson distribution: Poisson distribution as a limiting case of Binomial distribution, mean and variance of Poisson distribution.

Normal distribution: mean, variance, mode, median of Normal distribution.

MODULE-IV: CORRELATION AND REGRESSION (09)
Correlation- Karl Pearson's coefficient of correlation, rank correlation, repeated ranks. Regression: Lines of regression, regression coefficient, angle between two regression lines.

MODULE-V: TEST OF HYPOTHESIS (10)
Population, Sample, standard error; Test of significance: Null hypothesis, alternate hypothesis. Types of errors, level of significance.
Large sample tests: Test of hypothesis for single mean, difference between means, single proportion and difference between proportions. Small sample tests: Student's t-distribution, F-distribution and Chi-square distribution.

V. TEXT BOOKS:
1. Erwin Kreyszig, "*Advanced Engineering Mathematics*", John Wiley and Sons Publishers, 9th Edition, 2014.
2. B. S. Grewal, *Higher Engineering Mathematics*, 44/e, Khanna Publishers, 2017.

VI. REFERENCE BOOKS:
1. S. C. Gupta, V. K. Kapoor, "*Fundamentals of Mathematical Statistics*", S. Chand and Co., 10th Edition, 2000.
2. N.P. Bali and Manish Goyal, *A text book of Engineering Mathematics*, Laxmi Publications, Reprint, 2008.
3. Richard Arnold Johnson, Irwin Miller and John E. Freund, "*Probability and Statistics for Engineers*", Prentice Hall, 8th Edition, 2013.

VII. ELECTRONIC RESOURCES:
1. http://e4uhu.com/down/Applied/9th
2. https://toaz.info/32fa2f50-8490-42cf-9e6a-f50cb7ea9a5
3. http://www.mathworld.wolfram.com

VIII. MATERIAL ONLINE:
1. Course template
2. Tech-talk topics
3. Assignments
4. Open end experiments
5. Definition and terminology
6. Tutorial question bank
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. Early lecture readiness videos (ELRV)
11. Power point presentations

COURSE CONTENT

COMPUTER SYSTEM ARCHITECTURE								
III Semester: CSE / IT / CSIT / CSE (AI&ML) / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AECD04	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisites: There are no prerequisites to take this course								

I. COURSE OVERVIEW:

This course is designed to provide students with a deep understanding of the fundamental principles that govern the design and operation of computer systems. The Course covers the organization of computer systems, memory management, I/O management, and multiprocessor systems. The course forms the basis for advanced studies and research in areas such as computer engineering, and related disciplines.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The concepts of register transfer logic and arithmetic operations, instruction format, and instruction cycle.
II. The basic components of computer systems, functionality, and interactions with the components
III. Memory hierarchy, memory management, and I/O management.
IV. Pipelining and Multiprocessor techniques for the improvement of efficiency

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO1	Demonstrate a thorough understanding of the basic concepts and principles of computer system architecture.
CO2	Analyze different types of instruction sets and addressing modes.
CO3	Evaluate memory management techniques such as paging, segmentation, and virtual memory.
CO4	Compare different I/O techniques, including programmed I/O, interrupt driven I/O, and direct memory access (DMA)
CO5	Explore the implications of parallel processing and apply concepts of pipelining and parallelism to enhance system performance.

IV. COURSE CONTENT:

MODULE – I: REGISTER TRANSFER AND MICROOPERATIONS (10)

Register transfer, Bus, and memory transfers, Arithmetic microoperations, Logic microoperations, Shift microoperations, and Arithmetic logic shift unit. Computer arithmetic: Addition and subtraction, floating point arithmetic operations, decimal arithmetic unit.

MODULE – II: ORGANIZATION OF A COMPUTER (09)

Instruction codes, Computer registers, Computer instructions, Timing and control, Instruction cycle, Program Input-Output and Interrupt. Instruction formats, Addressing modes, Data Transfer and Manipulation, Program Control, RISC.

MODULE – III: MICROPROGRAMMED CONTROL AND INPUT-OUTPUT ORGANIZATION (10)

Micro Programmed Control: Control memory, Address sequencing, Design of control unit, Hardwired control, Micro programmed control.

Input-Output Organization: Peripheral devices, Input-Output interface, Modes of transfer, Priority interrupt – Daisy chaining priority, Parallel priority interrupt, Priority encoder; Direct Memory Access, Input-Output Processor – CPU-IOP communication; PCI Express - PCI physical and logical architecture.

MODULE - IV: MEMORY ORGANIZATION (09)

Memory organization: Memory hierarchy, main memory, auxiliary memory, associative memory, cache memory, virtual memory; Semiconductor RAMs – Internal organization, Static memories, Dynamic RAMs, Synchronous and Asynchronous DRAMs, Structure of larger memories; Read-only memories, Cache memories – Mapping functions; Nonvolatile Solid-State Memory Technologies, Solid state drives.

MODULE – V: MULTIPROCESSORS (09)

Pipeline and Vector Processing: Parallel processing, Pipelining, Instruction pipeline, Vector processing, Array processors. Multiprocessors: Characteristics of multiprocessors, Interconnection structures, Inter-processor arbitration. Multicore Computers: Hardware performance issues, Software performance issues, Multicore organization, Intel Core i7-990X.

V. TEXTBOOKS:

1. M. Morris Mano, "Computer Systems Architecture", Pearson, 3rd edition, 2015.
2. Patterson, Hennessy, "Computer Organization and Design: The Hardware/Software Interface", Morgan Kaufmann, 5th edition, 2013.

VI. REFERENCE BOOKS:

1. John. P. Hayes, "Computer System Architecture", McGraw-Hill, 3rd edition, 1998.
2. Carl Hamacher, Zvonko G Vranesic, Safwat G Zaky, "Computer Organization", McGraw-Hill, 5th edition, 2002.
3. William Stallings, "Computer Organization and Architecture", Pearson Edition, 8th edition, 2010.

VII. WEB REFERENCES:

1. https://www.tutorialspoint.com/computer_logical_organization/
2. https://www.courseera.org/learn/comparch
3. https://www.cssimplified.com/computer-organization-and-assembly-language-programming

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

OPERATING SYSTEMS

III Semester: CSE / CSE (CS) / CSE(DS) / CSE (AI & ML) / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSD09	Core	3	0	0	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite: Python Programming								

I. COURSE OVERVIEW:

Operating system course gives you skills and ways to think about common services for computer programs.
It is designed to provide an in-depth critique of the problems of resource management, scheduling, concurrency, synchronization, memory management, file management, peripheral management, protection, and security. It deals with the transfer of programs in and out of memory; and organizes processing time between programs and users. Learned knowledge will be implemented in the design and development of hybrid operating systems, command control systems, and real-time environments.

II. COURSE OBJECTIVES:

The students will try to learn

I. The principles of operating systems, services and functionalities with its evolution.
II. The structures, functions and components of modern operating systems
III. The conventional hardware at different OS abstraction levels.
IV. The essential skills to examine issues and methods employed in design of operating systems with identification of various functionalities.

III. COURSE OUTCOMES:

At the end of the course, students should be able to:

CO1	Demonstrate different architectures used in the design of modern operating systems.
CO2	Solve problems related to process scheduling, synchronization, and deadlock handling in uniprocessor and multi-processing systems.
CO3	Implement memory allocation algorithms for effective utilization of resources.
CO4	Select various page replacement algorithms applied for the allocation of frames...
CO5	Analyze different file allocation methods and disk scheduling algorithms applied for efficient utilization of storage.
CO6	Outline mechanisms used in the protection of resources in real-time environment

IV. COURSE CONTENT:

MODULE – I: INTRODUCTION (10)

Operating systems objectives and functions: Computer system architecture, operating systems structure, operating systems operations; Evolution of operating systems: Simple batch, multi programmed, time shared, personal computer, parallel distributed systems, real time systems, special purpose systems, operating system services, user operating systems interface; Systems calls: Types of systems calls, system programs, protection and security, operating system design and implementation, operating systems structure, virtual machines.

MODULE – II: PROCESS AND CPU SCHEDULING, PROCESS COORDINATION (09)

Process concepts: The process, process state, process control block, threads; Process scheduling: Scheduling queues, schedulers, context switch, preemptive scheduling, dispatcher, scheduling criteria, scheduling algorithms, multiple processor scheduling; Real time scheduling; Thread scheduling; Case studies Linux windows; Process synchronization, the critical section problem; Peterson‘s solution, synchronization hardware, semaphores and classic problems of synchronization, monitors

MODULE – III: MEMORY MANAGEMENT AND VIRTUAL MEMORY (10)
Logical and physical address space: Swapping, contiguous memory allocation, paging, structure of page table.

Segmentation: Segmentation with paging, virtual memory, demand paging; Performance of demand paging: Page replacement, page replacement algorithms, allocation of frames, thrashing.

MODULE – IV: FILE SYSTEM INTERFACE, MASS-STORAGE STRUCTURE (9)
The concept of a file, access methods, directory structure, file system mounting, file sharing, protection, file system structure, file system implementation, allocation methods, free space management, directory implementation, efficiency and performance; Overview of mass storage structure: Disk structure, disk attachment, disk scheduling, disk management, swap space management; Dynamic memory allocation: Basic concepts; Library functions.

MODULE –V: DEADLOCKS, PROTECTION (10)
System Model: Deadlock characterization, methods of handling deadlocks, deadlock prevention, dead lock avoidance, dead lock detection and recovery form deadlock system protection, goals of protection, principles of protection, domain of protection, access matrix, implementation of access matrix, access control, revocation of access rights, capability-based systems, language-based protection.

V. TEXT BOOKS:

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, “Operating System Principles”, Wiley Student Edition, 8th edition, 2010.
2. William Stallings, “Operating System- Internals and Design Principles”, Pearson Education, 6th edition, 2002.

VI. REFERENCE BOOKS:

1. Andrew S Tanenbaum, “Modern Operating Systems”, PHI, 3rd Edition, 2007.
2. D. M. Dhamdhere, “Operating Systems a Concept Based Approach”, Tata McGraw-Hill, 2nd Edition, 2006.

VII. ELECTRONICS RESOURCES:

1. www.smartzworld.com/notes/operatingsystems
2. www.scoopworld.in
3. www.sxecw.edu.in
4. www.technofest2u.blogspot.com

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DATA STRUCTURES

III Semester: AE / ME / CE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT / ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSD08	Core	3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		

Prerequisite: Essentials of Problem Solving

I. COURSE OVERVIEW:
The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSES OBJECTIVES:
The students will try to learn

I. The skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.
II. The basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.
III. The fundamentals of how to store, retrieve, and process data efficiently.
IV. The implementing these data structures and algorithms in Python.
V. The essential for future programming and software engineering courses.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO 1	Interpret the complexity of the algorithm using the asymptotic notations.
CO 2	Select the appropriate searching and sorting technique for a given problem
CO 3	Construct programs on performing operations on linear and nonlinear data structuresfor organization of a data
CO 4	Make use of linear data structures and nonlinear data structures solving real-time applications.
CO 5	Describe hashing techniques and collision resolution methods for accessing data with respect to performance
CO 6	Compare various types of data structures; in terms of implementation, operations and performance.

IV. COURSE CONTENT:
MODULE – I: INTRODUCTION TO DATA STRUCTURES, SEARCHING AND SORTING (09)
Basic concepts: Introduction to data structures, classification of data structures, operations on data structures, Algorithm Specification, Recursive algorithms, Data Abstraction, Performance analysis- time complexity and space complexity, Introduction to Linear and Non Linear data structures, Searching techniques: Linear and Binary search, Uniform Binary Search, Interpolation Search, Fibonacci Search; Sorting techniques: Bubble, Selection, Insertion, and Quick, Merge, Radix and Shell Sort and comparison of sorting algorithms.

MODULE – II: LINEAR DATA STRUCTURES (09)
Stacks: Stack ADT, definition and operations, Implementations of stacks using array, applications of stacks, Arithmetic expression conversion and evaluation; Queues: Primitive operations; Implementation of queues using

Arrays, applications of linear queue, circular queue and double ended queue (deque).

MODULE – III: LINKED LISTS (09)

Linked lists: Introduction, singly linked list, representation of a linked list in memory, operations on a single linked list; Applications of linked lists: Polynomial representation and sparse matrix manipulation.

Types of linked lists: Circular linked lists, doubly linked lists; Linked list representation and operations of Stack, linked list representation and operations of queue.

MODULE – IV: NON LINEAR DATA STRUCTURES (09)

Trees: Basic concept, binary tree, binary tree representation, array and linked representations, binary tree traversal, binary tree variants, threaded binary trees, application of trees, Graphs: Basic concept, graph terminology, Graph Representations

- Adjacency matrix, Adjacency lists, graph implementation, Graph traversals – BFS, DFS, Application of graphs, Minimum spanning trees – Prims and Kruskal algorithms.

MODULE – V: BINARY TREES AND HASHING (09)

Binary search trees: Binary search trees, properties and operations; Balanced search trees: AVL trees; Introduction to M- Way search trees, B trees; Hashing and collision: Introduction, hash tables, hash functions, collisions, applications of hashing.

V. TEXT BOOKS:

1. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.

VI. REFERENCE BOOKS:

1. S. Lipschutz, "Data Structures", Tata McGraw Hill Education, 1st Edition, 2008.
2. D. Samanta, "Classic Data Structures", PHI Learning, 2nd Edition, 2004.

VII. ELECTRONICS RESOURCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Definition and terminology
4. Tech-talk topics
5. Assignments
6. Model question paper - I
7. Model question paper - II
8. Lecture notes
9. Early learning readiness videos (ELRV)
10. Power point presentations

COURSE CONTENT

MATHEMATICS FOR COMPUTING								
III Semester: CSE / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AITD01	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		
Prerequisite: Python Programming								

I. COURSE OVERVIEW:

Computer science depends up on the science of mathematics, in order to acquire the knowledge in computing, mathematical ideas are required. Course is to provide a clear understanding of the concepts that underlying fundamentals with emphasis on their applications to computer science. It highlights mathematical definitions and proofs as well as applicable methods. The contents include formal logic notation, proof methods; induction, well-ordering; sets, relations; growth of functions; permutations and combinations, counting principles, recurrence equations.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental knowledge of statement notations and logical connectives which are used to convert English sentences into logical expressions.
II. The effective use of combinatory principles for calculating probabilities and solving counting problems
III. Relate practical examples to the functions and relations and interpret the associated operations and terminology used in the context
IV. The characteristics of generating functions for finding the solution of linear homogeneous recurrence relations

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Make use of Number system and converting Decimal to binary, octal and hexadecimal and also gray code to binary, Binary to gray code..
CO2	Demonstrate notations for reformulating statements in formal logic and validating normal forms.
CO3	Demonstrate operations on discrete mathematical structures like sets, functions, lattices for representing the relations among them.
CO4	Illustrate rings, integral domains, and field structures with binary operations defined on them. .
CO5	Apply addition rule and substitution rule for solving the problems of combinatory
CO6	Develop solutions for recurrence relations and generating functions to obtain terms of equations.

IV. COURSE CONTENT:

MODULE – I: NUMBER SYSYTEM (10)

Number Systems: Basics, Numbers in base 10, The Binary System, Calculating the system, Octal number system, Hexa Decimal number system, Converting Decimal to Binary, Octal and Hexadecimal System. Gray code, Converting Gray code to Binary and Binary to Gray code.

MODULE – II: MATHEMATICAL LOGIC (10)

Propositional logic and Predicate Calculus: Statements and Notations, Connectives, Truth Tables, Tautologies, Equivalence of Formulas, Tautological Implications, Normal Forms, Theory of Inference for Statement Calculus, Consistency of Premises, Indirect Method of Proof, Predicative Logic, Statement Functions,

MODULE – III: RELATIONS, FUNCTIONS AND LATTICES (10)
Introduction to Sets, representation of Sets, Operation on Sets, Properties of Binary Relations, Relation Matrix, Operations on Relations, Transitive Closure, Equivalence Relation, Compatibility and Partial Ordering Relations, Hasse Diagrams, Lattices: LUB, GLB. Functions: Bijective Functions, Composition of Functions, Inverse Functions,

MODULE – IV: ALGEBRAIC STRUCTURES AND COMBINATORICS (09)
Algebraic structures: Algebraic systems, examples and general properties, semi groups and monoids, groups, sub groups, homomorphism, isomorphism, rings.
Combinatory: The fundamental counting principles, permutations, disarrangements, combinations, permutations and combinations with repetitions, the binomial theorem, multinomial theorem, generalized inclusion exclusion principle.

MODULE – V: RECURRENCE RELATION (09)
Recurrence relation: Generating functions, function of sequences calculating coefficient of generating function, recurrence relations, solving recurrence relation by substitution and generating functions, Characteristics roots solution of homogeneous recurrence relation.

V. TEXT BOOKS:

1. J. P. Tremblay, R. Manohar, “Discrete Mathematical Structures with Applications to Computer Science”, Tata McGraw Hill, India, 1st edition, 1997.
2. JoeL. Mott, Abraham Kandel, Theodore P. Baker, “Discrete Mathematics for Computer Scientists and Mathematicians”, Prentice Hall of India Learning Private Limited, New Delhi, India, 2nd edition, 2010.

VI. REFERENCE BOOKS:

1. Kenneth H. Rosen, “Discrete Mathematics and Its Applications”, Tata Mcgraw-Hill, New Delhi, India, 6th edition, 2012.
2. C. L. Liu, D. P. Mohapatra, “Elements of Discrete Mathematics”, Tata Mcgraw-Hill, India, 3rd edition, 2008.
3. Ralph P. Grimaldi, B. V. Ramana, “Discrete and Combinatorial Mathematics - An Applied Introduction”, Pearson Education, India, 5th edition, 2011.
4. D. S. Malik, M. K. Sen, “Discrete Mathematical Structures: Theory and Applications”, Thomson Course Technology, India, 1st edition, 2004.

VII. ELECTRONIC RESOURCES:

1. https://swayam.gov.in/explorer?searchText=Discrete+Mathematical+Structures
2. https://www.javatpoint.com/discrete-mathematics-tutorial.
3. http://www.web.stanford.edu/class/cs103x
4. http://www.cs.odu.edu/~cs381/cs381content/web_course.html
5. http://www.cse.iitd.ernet.in/~bagchi/courses/discrete-book
6. http://www.nptel.ac.in/courses/106106094/
7. http://www.tutorialspoint.com/discrete_mathematics
8. http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs.

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

OPERATING SYSTEMS LABORATORY								
III Semester: CSE / IT / CSE (AI&ML) / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD10	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:
The course covers some of the design aspects of operating system concepts. Topics covered include process scheduling, memory management, deadlocks, disk scheduling strategies, and file allocation methods. The main objective of the course is to teach the students how to select and design algorithms that are appropriate for problems that they might encounter in real life.

II. COURSE OBJECTIVES
The students will try to learn:

I. The functionalities of main components in operating systems and analyze the algorithms used in process management.
II. Algorithms used in memory management and I/O management
III. Different methods for preventing or avoiding deadlocks and File systems.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO1	Acquire knowledge of the operating system structure and process
CO2	Analyze the performance of process scheduling algorithms
CO3	Evaluate the process memory requirement and its fragmentation.
CO4	Analyze the safe state and deadlock mechanism
CO5	Analyze the performance of the disk scheduling algorithms
CO6	Ability to simulate file structures and allocation methods

IV. COURSE CONTENT:

OPERATING SYSTEMS LABORATORY COURSE CONTENT

F

IV. COURSE CONTENT:

EXERCISES FOR OPERATING SYSTEMS LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 The Busy Printer

Imagine a IARE computer lab with a single printer that serves multiple students. The lab operates on a first-come, first-served basis for print jobs.

One morning, the lab opens, and the printer is ready to accept jobs. The following print jobs arrive in sequence:

Job A: Submitted by A, consisting of 10 pages.

Job B: Submitted by B, consisting of 5 pages.

Job C: Submitted by C, consisting of 3 pages.

Job D: Submitted by D, consisting of 7 pages.

Each job arrives at the lab at different times, and they start printing immediately upon arrival. The printer processes each job in the order it arrives, and each job completes printing without interruption.

Your task is to simulate the FCFS scheduling algorithm for these print jobs and calculate the total time taken to complete all printing tasks, assuming that the printer prints at a rate of 1 page per second.

Input: The arrival time of each job (in seconds). The number of pages to be printed for each job.

Job A: Arrival time = 0 seconds, Pages = 10

Job B: Arrival time = 1 second, Pages = 5

Job C: Arrival time = 3 seconds, Pages = 3

Job D: Arrival time = 5 seconds, Pages = 7

Output: The total time (in seconds) taken to complete all printing tasks.

27 seconds

Assumptions:

The printer starts immediately on each job's arrival.

No additional delay between jobs.

Explanation:

Job A arrives first at time 0 and takes 10 seconds to print (10 pages).

Job B arrives at time 1 and starts printing immediately after Job A completes. It takes 5 seconds to print (5 pages).

Job C arrives at time 3 and starts immediately after Job B completes. It takes 3 seconds to print (3 pages).

Job D arrives at time 5 and starts immediately after Job C completes. It takes 7 seconds to print (7 pages).

Thus, the total time taken to complete all print jobs is 27 seconds.

```
def calculate_total_time(arrival_times, pages):
    current_time = 0
    total_time = 0
    for i in range(len(arrival_times)):
        # Write code here

# Driver Code
arrival_times = [0, 1, 3, 5]
pages = [10, 5, 3, 7]

total_time = calculate_total_time(arrival_times, pages)

print(f"The total time taken to complete all printing tasks is: {total_time} seconds")
```

1.2 The Software Developer's Tasks

Sophia, a software developer, has several programming tasks to complete in her project. Each task requires a different amount of time to finish. She decides to use the SJF scheduling algorithm to manage her tasks efficiently.
Here are the tasks she needs to complete:

- **Task A:** Writing a simple utility function, which takes 3 hours.
- **Task B:** Debugging a critical issue in the main application, which takes 5 hours.
- **Task C:** Refactoring a complex module, which takes 7 hours.
- **Task D:** Testing and documenting a new feature, which takes 4 hours.

Sophia decides to implement the SJF algorithm for scheduling these tasks based on their execution times.

Your task is to simulate the SJF scheduling algorithm for Sophia's tasks and calculate the total time required to complete all tasks, assuming that each task starts immediately after the previous one is completed.

Input: Execution times of each task.
Task A: 3 hours
Task B: 5 hours
Task C: 7 hours
Task D: 4 hours

Output: The total time (in hours) required to complete all tasks.
19 hours

Assumptions: The tasks are executed in the order of their shortest to longest execution times using SJF.
Each task starts immediately after the previous one is completed.
Explanation:

Using SJF, the tasks will be scheduled in the following order:

Task A (3 hours)

Task D (4 hours)

Task B (5 hours)

Task C (7 hours)

Thus, the total time required to complete all tasks is 19 hours.

```
def calculate_total_time(execution_times):
    # Write code here
    return total_time

# Driver Code
```

```
execution_times = [3, 5, 7, 4]
total_time = calculate_total_time(execution_times)

print(f"The total time required to complete all tasks is: {total_time} hours")
```

1.3 Managing a Restaurant's Orders

Imagine you are managing a busy restaurant where orders from different tables need to be processed. Each order requires a different amount of time to be prepared and served. To efficiently manage the kitchen staff and ensure timely service, you decide to implement the Round Robin scheduling algorithm for processing orders.

Here are the details of the current orders:

Order from Table 1: Burger and Fries, which takes 5 minutes to prepare.

Order from Table 2: Salad, which takes 3 minutes to prepare.

Order from Table 3: Pizza, which takes 8 minutes to prepare.

Order from Table 4: Pasta, which takes 6 minutes to prepare.

You decide to implement Round Robin with a time quantum of 4 minutes. This means each order will receive processing time in chunks of 4 minutes until it is completed or until it reaches its final processing time.

Your task is to simulate the Round Robin scheduling algorithm for processing these orders and calculate the total time required to complete all orders, considering the time quantum.

Input: Processing times of each order. Time quantum for Round Robin scheduling.

Order from Table 1: Burger and Fries (5 minutes)

Order from Table 2: Salad (3 minutes)

Order from Table 3: Pizza (8 minutes)

Order from Table 4: Pasta (6 minutes)

Time Quantum: 4 minutes

Output: The total time (in minutes) required to complete all orders.

22 minutes

Assumptions:

Each order starts processing immediately.

If an order cannot be completed within one quantum, it will continue in subsequent rounds.

Process orders in the order they were received.

Explanation:

Round 1:

Table 1 (Burger and Fries) - Processed for 4 minutes.

Table 2 (Salad) - Processed for 3 minutes (completed).

Table 3 (Pizza) - Processed for 4 minutes.

Table 4 (Pasta) - Processed for 4 minutes.

Round 2:

Table 1 (Burger and Fries) - Processed for remaining 1 minute.

Table 3 (Pizza) - Processed for remaining 4 minutes.

Table 4 (Pasta) - Processed for remaining 2 minutes (completed).

Thus, the total time required to complete all orders using Round Robin scheduling with a time quantum of 4 minutes is 22 minutes.

```python
from collections import deque

def calculate_total_time(orders, time_quantum):
    # Write code here
    return total_time

# Driver Code
orders = [5, 3, 8, 6]
time_quantum = 4
total_time = calculate_total_time(orders, time_quantum)
print(f"The total time required to complete all orders is: {total_time} minutes")
```

1.4 Emergency Room Prioritization

You are managing an emergency room in a hospital where patients arrive with different medical conditions. Each patient requires a different level of urgency for treatment. To ensure critical cases are handled promptly, you decide to implement the Priority Scheduling algorithm for patient treatment.

Here are the details of the patients currently in the emergency room:

Patient A: Has suffered a heart attack and requires immediate attention, priority level 1.

Patient B: Has a broken arm and needs urgent treatment, priority level 2.

Patient C: Has a high fever and needs attention soon, priority level 3.

Patient D: Has a minor cut and can wait, priority level 4.

You decide to implement Priority Scheduling where patients with higher priority levels are treated first. If two patients have the same priority level, they are treated in the order they arrived.

Your task is to simulate the Priority Scheduling algorithm for treating these patients and calculate the total time required to treat all patients, considering their priority levels.

Input: Priority levels of each patient. Treatment times for each patient.

Patient A: Priority 1, Treatment time 10 minutes

Patient B: Priority 2, Treatment time 8 minutes

Patient C: Priority 3, Treatment time 15 minutes

Patient D: Priority 4, Treatment time 5 minutes

Output: The total time (in minutes) required to treat all patients.

38 minutes

Assumptions: Each patient starts treatment immediately upon arrival.

Patients are treated based on their priority levels (lower number = higher priority).

If two patients have the same priority level, they are treated in the order they arrived.

Explanation:

Patients are treated in the following order based on their priority:

Patient A (Priority 1, 10 minutes)

Patient B (Priority 2, 8 minutes)

Patient C (Priority 3, 15 minutes)

Patient D (Priority 4, 5 minutes)

Thus, the total time required to treat all patients using Priority Scheduling is 38 minutes.

```
from queue import PriorityQueue

def calculate_total_time(patients):
    # Write code here
    return total_time

# Drive Code
patients = [(1, 10),  # Patient A: Priority 1, Treatment time 10 minutes
            (2, 8),   # Patient B: Priority 2, Treatment time 8 minutes
            (3, 15),  # Patient C: Priority 3, Treatment time 15 minutes
            (4, 5)    # Patient D: Priority 4, Treatment time 5 minutes
           ]

total_time = calculate_total_time(patients)
print(f"The total time required to treat all patients is: {total_time} minutes")
```

2. Multi-Level Queue Scheduling

2.1 Managing System and User Processes

You are managing a computer system that runs various types of processes, categorized into system processes and user processes. System processes are critical for the functioning of the operating system and are given higher priority compared to user processes. To efficiently manage these processes, you decide to implement a Multi-Level Queue Scheduling algorithm with FCFS scheduling for each queue.
Here's the breakdown of the processes currently in the system:

System Process Queue:
Process A: Handles file system operations, takes 5 units of time to complete.
Process B: Manages memory allocation, takes 3 units of time to complete.
Process C: Performs system updates, takes 7 units of time to complete.

User Process Queue:
Process D: Runs a word processor, takes 4 units of time to complete.
Process E: Executes a media player, takes 2 units of time to complete.
Process F: Performs data analysis, takes 6 units of time to complete.

Implement the Multi-Level Queue Scheduling algorithm where system processes are scheduled with higher priority compared to user processes. Within each queue (system and user), processes are scheduled using FCFS.

Your task is to simulate the scheduling of these processes and calculate the total time required to complete all processes, considering the priorities and FCFS scheduling within each queue.

Input: Processing times for each process in the system and user queues.
System Processes:
Process A: 5 units
Process B: 3 units
Process C: 7 units

User Processes:
Process D: 4 units
Process E: 2 units
Process F: 6 units

Output: The total time (in units) required to complete all processes.
27 units

Assumptions: Each process starts execution immediately upon arrival. System processes have higher priority over user processes. FCFS scheduling is applied within each queue (system and user).

Explanation:
System Process Queue:
Process A (5 units)
Process B (3 units)
Process C (7 units)

User Process Queue:
Process D (4 units)
Process E (2 units)
Process F (6 units)
The total time required to complete all processes using Multi-Level Queue Scheduling with FCFS within each queue is 27 units.

```
from queue import Queue

def calculate_total_time(system_processes, user_processes):
    # Write code here
    return total_time

# Driver Code
system_processes = {'Process A': 5,'Process B': 3,'Process C': 7,}

user_processes = {'Process D': 4, 'Process E': 2, 'Process F': 6,}

total_time = calculate_total_time (system_processes, user_processes)
print(f"The total time required to complete all processes is: {total_time} units")
```

2.2 Managing Job Scheduling in a Computing Center

You are managing a computing center that processes jobs submitted by various departments of an organization. Jobs are categorized into two priority levels: high priority (system jobs) and low priority (user jobs). Each job has a specific processing time required for completion.

Here are the details of the jobs currently in the system:

High Priority (System) Jobs:
Job A: Performs critical database backups, takes 8 units of time to complete.
Job B: Handles real-time monitoring tasks, takes 5 units of time to complete.
Job C: Executes system updates, takes 10 units of time to complete.

Low Priority (User) Jobs:
Job D: Runs periodic report generation, takes 6 units of time to complete.
Job E: Processes email delivery, takes 3 units of time to complete.
Job F: Executes batch data processing, takes 7 units of time to complete.

Implement the Multi-Level Queue Scheduling algorithm where high priority (system) jobs are scheduled with higher priority compared to low priority (user) jobs. Within each queue (high and low priority), jobs are scheduled using FCFS.

Your task is to simulate the scheduling of these jobs and calculate the total time required to complete all jobs, considering the priorities and FCFS scheduling within each queue.

Input: Processing times for each job in the high and low priority queues.
High Priority Jobs:
Job A: 8 units
Job B: 5 units
Job C: 10 units

Low Priority Jobs:
Job D: 6 units
Job E: 3 units
Job F: 7 units

Output: The total time (in units) required to complete all jobs.
39 units
Assumptions: Each job starts execution immediately upon arrival. High priority (system) jobs have higher priority over low priority (user) jobs. FCFS scheduling is applied within each queue (high and low priority).

Explanation:
High Priority (System) Jobs:
Job A (8 units)
Job B (5 units)
Job C (10 units)

Low Priority (User) Jobs:
Job D (6 units)
Job E (3 units)
Job F (7 units)

The total time required to complete all jobs using Multi-Level Queue Scheduling with FCFS within each queue is 39 units.

```
from queue import Queue

def calculate_total_time(high_priority_jobs, low_priority_jobs):
    # Write code here
    return total_time

# Write code here
high_priority_jobs = {'Job A': 8, 'Job B': 5, 'Job C': 10,}

low_priority_jobs = {'Job D': 6, 'Job E': 3, 'Job F': 7,}

total_time = calculate_total_time(high_priority_jobs, low_priority_jobs)

print(f"The total time required to complete all jobs is: {total_time} units")
```

2.3 Managing Print Jobs in a Shared Printing Environment

You are managing a shared printing environment where print jobs from different departments are queued up for processing. Print jobs are categorized into two priority levels: high priority (system jobs) and low priority (user jobs). Each job has a specific processing time required for printing.

Here are the details of the print jobs currently in the system:

High Priority (System) Print Jobs:
Job A: Urgent report for management, takes 15 units of time to print.
Job B: Critical financial statement, takes 10 units of time to print.
Job C: Emergency document for legal department, takes 20 units of time to print.
Low Priority (User) Print Jobs:
Job D: Regular office memo, takes 5 units of time to print.
Job E: Marketing flyer, takes 8 units of time to print.
Job F: Personal document for an employee, takes 3 units of time to print.

Implement the Multi-Level Queue Scheduling algorithm where high priority (system) print jobs are scheduled with higher priority compared to low priority (user) print jobs. Within each queue (high and low priority), print jobs are scheduled using FCFS.

Your task is to simulate the scheduling of these print jobs and calculate the total time required to complete all print jobs, considering the priorities and FCFS scheduling within each queue.

Input: Printing times for each job in the high and low priority queues.
High Priority Print Jobs:
Job A: 15 units
Job B: 10 units
Job C: 20 units

Low Priority Print Jobs:
Job D: 5 units
Job E: 8 units
Job F: 3 units

Output: The total time (in units) required to complete all print jobs.
61 units

Assumptions: Each print job starts printing immediately upon arrival. High priority (system) print jobs have higher priority over low priority (user) print jobs. FCFS scheduling is applied within each queue (high and low priority).

Explanation:

High Priority (System) Print Jobs:
Job A (15 units)
Job B (10 units)
Job C (20 units)

Low Priority (User) Print Jobs:
Job D (5 units)
Job E (8 units)
Job F (3 units)

The total time required to complete all print jobs using Multi-Level Queue Scheduling with FCFS within each queue is 61 units.

```
from queue import Queue

def calculate_total_time(high_priority_jobs, low_priority_jobs):
```

```python
    # Write code here
    return total_time

# Driver Code
high_priority_jobs = {'Job A': 15, 'Job B': 10, 'Job C': 20,}

low_priority_jobs = {'Job D': 5, 'Job E': 8, 'Job F': 3,}

total_time = calculate_total_time(high_priority_jobs, low_priority_jobs)

print(f"The total time required to complete all print jobs is: {total_time} units")
```

2.4 Task Scheduling in a Multi-User System

You are managing a multi-user system where tasks from different users need to be scheduled for processing. Tasks are categorized into two priority levels: high priority (system tasks) and low priority (user tasks). Each task has a specific processing time required for completion.

Here are the details of the tasks currently in the system:
High Priority (System) Tasks:
Task A: Performs critical system maintenance, takes 12 units of time to complete.
Task B: Executes essential security updates, takes 8 units of time to complete.
Task C: Handles real-time data processing, takes 15 units of time to complete.
Low Priority (User) Tasks:
Task D: Runs a background data synchronization process, takes 6 units of time to complete.
Task E: Processes user-generated reports, takes 4 units of time to complete.
Task F: Executes regular data backups, takes 10 units of time to complete.

Implement the Multi-Level Queue Scheduling algorithm where high priority (system) tasks are scheduled with higher priority compared to low priority (user) tasks. Within each queue (high and low priority), tasks are scheduled using FCFS.

Your task is to simulate the scheduling of these tasks and calculate the total time required to complete all tasks, considering the priorities and FCFS scheduling within each queue.

Input: Processing times for each task in the high and low priority queues.
High Priority Tasks:
Task A: 12 units
Task B: 8 units
Task C: 15 units

Low Priority Tasks:
Task D: 6 units
Task E: 4 units
Task F: 10 units

Output: The total time (in units) required to complete all tasks.
55 units

Assumptions: Each task starts processing immediately upon arrival. High priority (system) tasks have higher priority over low priority (user) tasks. FCFS scheduling is applied within each queue (high and low priority).

Explanation:
High Priority (System) Tasks:
Task A (12 units)

Task B (8 units)
Task C (15 units)

Low Priority (User) Tasks:
Task D (6 units)
Task E (4 units)
Task F (10 units)

The total time required to complete all tasks using Multi-Level Queue Scheduling with FCFS within each queue is 55 units.

```
from queue import Queue

def calculate_total_time(high_priority_tasks, low_priority_tasks):
    # Write code here
    return total_time

# Driver Code
high_priority_tasks = {'Task A': 12, 'Task B': 8, 'Task C': 15,}

low_priority_tasks = {'Task D': 6, 'Task E': 4, 'Task F': 10, }

total_time = calculate_total_time(high_priority_tasks, low_priority_tasks)

print(f"The total time required to complete all tasks is: {total_time} units")
```

2.5 Job Scheduling in a Computing Cluster

You are managing a computing cluster that processes jobs submitted by various departments of a research institution. Jobs are categorized into three priority levels: high priority (critical jobs), medium priority (standard jobs), and low priority (background jobs). Each job has a specific processing time required for completion.

Here are the details of the jobs currently in the system:

High Priority (Critical) Jobs:
Job A: Performs complex simulations for a time-sensitive research project, takes 20 units of time to complete.
Job B: Analyzes large datasets for an urgent analysis, takes 15 units of time to complete.
Job C: Executes critical experiments with strict deadlines, takes 25 units of time to complete.

Medium Priority (Standard) Jobs:
Job D: Runs routine data processing tasks, takes 10 units of time to complete.
Job E: Processes experimental results for ongoing studies, takes 12 units of time to complete.
Job F: Executes regular system maintenance tasks, takes 8 units of time to complete.

Low Priority (Background) Jobs:
Job G: Performs non-critical data backups, takes 5 units of time to complete.
Job H: Runs periodic log file analysis, takes 4 units of time to complete.
Job I: Executes routine software updates, takes 6 units of time to complete.

Implement the Multi-Level Queue Scheduling algorithm where jobs are scheduled based on their priority levels: high priority jobs are scheduled first, followed by medium priority jobs, and then low priority jobs. Within each priority level, jobs are scheduled using FCFS.

Your task is to simulate the scheduling of these jobs and calculate the total time required to complete all jobs, considering the priorities and FCFS scheduling within each priority level.

Input: Processing times for each job in the high, medium, and low priority queues.
High Priority Jobs:
Job A: 20 units
Job B: 15 units
Job C: 25 units

Medium Priority Jobs:
Job D: 10 units
Job E: 12 units
Job F: 8 units

Low Priority Jobs:
Job G: 5 units
Job H: 4 units
Job I: 6 units

Output: The total time (in units) required to complete all jobs.
105 units

Assumptions: Each job starts execution immediately upon arrival. Jobs within higher priority levels have higher priority over jobs in lower priority levels. FCFS scheduling is applied within each priority level.

Explanation:
High Priority Jobs:
Job A (20 units)
Job B (15 units)
Job C (25 units)

Medium Priority Jobs:
Job D (10 units)
Job E (12 units)
Job F (8 units)
Low Priority Jobs:
Job G (5 units)
Job H (4 units)
Job I (6 units)

The total time required to complete all jobs using Multi-Level Queue Scheduling with FCFS within each priority level is 105 units.

```
from queue import Queue

def calculate_total_time(high_priority_jobs, medium_priority_jobs, low_priority_jobs):
    # Write code here
    return total_time

# Driver Code
high_priority_jobs = {'Job A': 20, 'Job B': 15, 'Job C': 25,}

medium_priority_jobs = {'Job D': 10, 'Job E': 12, 'Job F': 8,}

low_priority_jobs = {'Job G': 5, 'Job H': 4, 'Job I': 6,}
```

```
total_time = calculate_total_time(high_priority_jobs, medium_priority_jobs,
low_priority_jobs)

print(f"The total time required to complete all jobs is: {total_time} units")
```

3. File Allocation Strategies

3.1 Managing Student Records in a School Database

You are tasked with designing a file management system for a school's student database. The database contains records of students' personal information, academic performance, and attendance history. Each student record is stored as a file with specific attributes.

Implement the Sequential File Allocation strategy to manage the storage and access of student records in the school database.
Here's the scenario of the school database system:

The school has records for three students:
Student A: John Doe, ID 101, Grade 10, Address: 123 Main Street
Student B: Jane Smith, ID 102, Grade 11, Address: 456 Elm Street
Student C: Michael Brown, ID 103, Grade 9, Address: 789 Oak Avenue

Implement and simulate the Sequential File Allocation strategy for storing and accessing these student records in the school database. Consider the following aspects:

File Structure: Each student record contains fields for Name, ID, Grade, and Address.
Storage: Files are stored sequentially one after another in contiguous blocks of disk space.
Access: A file allocation table (FAT) keeps track of the starting block address and the length of each student record file.

Your task is to design a data structure or representation for the student records. Implement the Sequential File Allocation strategy to store and retrieve student records. Calculate the total disk space used and evaluate the efficiency of the strategy based on storage utilization and access times.

Assumptions: Disk space is divided into fixed-size blocks. Each student record is stored in whole blocks (no partial blocks).

The Sequential File Allocation strategy should handle new record insertions, deletions, and updates efficiently, considering disk fragmentation and access patterns. To simulate the Sequential File Allocation strategy for managing student records in a school database, we'll create a Python program. In this program, we'll represent each student record as a file with fields for Name, ID, Grade, and Address. We'll use a list to simulate disk blocks and a file allocation table (FAT) to keep track of file locations.

```
class StudentRecord:
    def __init__(self, name, student_id, grade, address):
        self.name = name
        self.student_id = student_id
        self.grade = grade
        self.address = address

    def __str__(self):
        return f"Name: {self.name}, ID: {self.student_id}, Grade: {self.grade},
Address: {self.address}"
```

```
class SequentialFileAllocation:
    def __init__(self, block_size):
        self.disk_blocks = []
        self.fat = {}
        self.block_size = block_size
        self.next_free_block = 0

    def add_record(self, student_record):
        # Write code here
        return True

    def get_record(self, student_id):
        # Write code here
        return None

    def calculate_record_size(self, student_record):
         # Write code here
        return 1

    def print_disk_status(self):
        # Write code here

# Driver Code
block_size = 1
file_system = SequentialFileAllocation(block_size)
student_records = [
        StudentRecord ("John Doe", 101, 10, "123 Main Street"),
        StudentRecord ("Jane Smith", 102, 11, "456 Elm Street"),
        StudentRecord ("Michael Brown", 103, 9, "789 Oak Avenue")
    ]
for record in student_records:
    file_system.add_record (record)

file_system.print_disk_status()

student_id = 102
retrieved_record = file_system.get_record (student_id)
if retrieved_record:
   print(f"\n Retrieved Record for ID {student_id}:")
   print(retrieved_record)
else:
   print(f"\n Record with ID {student_id} not found.")
```

3.2 Managing Medical Records in a Hospital Information System

You have been tasked with designing a file management system for a hospital's electronic medical records (EMR). The EMR system contains detailed records of patients' medical histories, treatments, and diagnostic results. Each patient record is stored as a file with specific attributes.

Implement the Indexed File Allocation strategy to efficiently manage the storage and access of patient records in the hospital's EMR system.

Here's the scenario of the hospital information system:

The hospital manages records for several patients:
Patient A: John Smith, Age 45, Medical ID 1001, Address: 123 Hospital Road
Patient B: Jane Doe, Age 32, Medical ID 1002, Address: 456 Clinic Avenue
Patient C: Michael Johnson, Age 58, Medical ID 1003, Address: 789 Medical Plaza

Implement and simulate the Indexed File Allocation strategy for storing and accessing these patient records in the hospital's EMR system. Consider the following aspects:

File Structure: Each patient record contains fields for Name, Age, Medical ID, and Address.
Storage: Each patient record has its own index block that contains pointers to the actual blocks of disk space where its data is stored.
Index Table: Maintain an index table that maps each patient record to its respective index block, facilitating efficient access.
Access: Support operations such as insertion, deletion, and retrieval of patient records based on their Medical ID.

Your task is to design a data structure or representation for patient records and index blocks. Implement the Indexed File Allocation strategy to store and retrieve patient records. Calculate the total disk space used and evaluate the efficiency of the strategy based on storage utilization and access times.
Assumptions: Disk space is divided into fixed-size blocks. Each patient record is stored in whole blocks (no partial blocks). The Indexed File Allocation strategy should handle new record insertions, deletions, and updates efficiently, considering disk fragmentation and access patterns.

To implement the Indexed File Allocation strategy for managing patient records in a hospital's electronic medical records (EMR) system, we'll create a Python program. This program will simulate the storage and retrieval of patient records using an index table and index blocks to efficiently manage disk space and access.

```
class PatientRecord:
    def __init__(self, name, age, medical_id, address):
        self.name = name
        self.age = age
        self.medical_id = medical_id
        self.address = address

    def __str__(self):
        return f"Name: {self.name}, Age: {self.age}, Medical ID: {self.medical_id}, 
Address: {self.address}"

class IndexedFileAllocation:
    def __init__(self, block_size):
        self.disk_blocks = []
        self.index_table = {}
        self.index_block_size = block_size
        self.next_free_block = 0

    def add_record(self, patient_record):
        # Write code here
        return True

    def get_record(self, medical_id):
        # Write code here
        return None

    def calculate_record_size(self, patient_record):
        # Write code here
        return 1

    def print_disk_status(self):
        # Write code here
# Driver Code
index_block_size = 1
file_system = IndexedFileAllocation(index_block_size)

patient_records = [
        PatientRecord("John Smith", 45, 1001, "123 Hospital Road"),
        PatientRecord("Jane Doe", 32, 1002, "456 Clinic Avenue"),
        PatientRecord("Michael Johnson", 58, 1003, "789 Medical Plaza")
    ]
for record in patient_records:
    file_system.add_record(record)

file_system.print_disk_status()

medical_id = 1002
retrieved_record = file_system.get_record(medical_id)
if retrieved_record:
   print(f"\nRetrieved Record for Medical ID {medical_id}:")
   print(retrieved_record)
else:
   print(f"\nRecord with Medical ID {medical_id} not found.")
```

3.3 Managing Digital Media Files in a Multimedia Application

You are developing a multimedia application that allows users to manage and store various types of digital media files, including images, videos, and audio clips. Each file is treated as a separate entity with its own attributes and content.

Implement the Linked File Allocation strategy to efficiently manage the storage and access of digital media files in your multimedia application.

Here's the scenario of the multimedia application:
The application needs to manage the following digital media files:

File A: Landscape.jpg (Image file), size 5 MB
File B: Concert.mp4 (Video file), size 50 MB
File C: Song.mp3 (Audio file), size 8 MB

Implement and simulate the Linked File Allocation strategy for storing and accessing these digital media files in the multimedia application. Consider the following aspects:

File Structure: Each digital media file has specific attributes (e.g., file name, type, size) and content.
Storage: Files are stored in non-contiguous blocks of disk space, and each block contains a pointer to the next block of the file.

File Allocation Table (FAT): Maintain a FAT to keep track of the starting block address of each file.
Access: Support operations such as insertion, deletion, and retrieval of files based on their attributes.

Your task is to design a data structure or representation for digital media files and disk blocks. Implement the Linked File Allocation strategy to store and retrieve digital media files. Calculate the total disk space used and evaluate the efficiency of the strategy based on storage utilization and access times.

Assumptions: Disk space is divided into blocks of fixed size. Each digital media file is stored in multiple blocks using pointers to link consecutive blocks. The Linked File Allocation strategy should handle new file insertions, deletions, and updates efficiently, considering disk fragmentation and access patterns.

```python
from threading import Semaphore

class DiskBlock:
    def __init__(self, block_id):
        self.block_id = block_id
        self.next_block = None


class MediaFile:
    def __init__(self, file_name, file_type, size):
        self.file_name = file_name
        self.file_type = file_type
        self.size = size
        self.start_block = None  # Points to the first block in the chain

    def __str__(self):
        return f"{self.file_name} ({self.file_type}), Size: {self.size} MB"


class FileAllocationTable:
    def __init__(self):
        self.fat = {}

    def allocate_blocks(self, media_file, disk_blocks):
        # Write code here

    def delete_file(self, media_file, disk_blocks):
         # Write code here

def simulate_linked_file_allocation():
    # Write code here

# Driver Code
simulate_linked_file_allocation()
```

3.4 DigitalArchive

You are a software engineer working for a digital library company called "DigitalArchive". DigitalArchive specializes in storing and managing vast collections of digital media files for various clients, including museums, libraries, and private collectors. Your task is to design a file allocation strategy that efficiently manages these files on DigitalArchive's storage system.

DigitalArchive has recently signed a contract with a renowned museum that wants to digitize and archive its extensive collection of historical photographs. The museum's collection includes thousands of photographs ranging from early 20th-century portraits to landscapes and architectural images.

Requirements:
File Types: Each photograph is stored as a digital image file (e.g., JPG format).
File Sizes: The sizes of photographs vary, with average file sizes ranging from 5 MB to 20 MB.
Storage System: DigitalArchive uses a storage system that organizes files into blocks of fixed size (e.g., 10 MB per block).
File Allocation Strategy: Sequential File Allocation

Sequential Allocation: In this strategy, files are stored sequentially on the storage system. When a new file arrives, it is allocated blocks of disk space one after another. Each file is assigned contiguous blocks of the required size.

Tasks to Implement: Design a program or algorithm that simulates the sequential file allocation strategy for storing the museum's photograph collection. Implement functions to allocate blocks for new photographs, manage the allocation of disk space, and retrieve information about allocated files. Ensure the program handles scenarios such as adding new photographs, calculating total disk space used, and displaying the allocation status.

Assume the storage system has 100 blocks of 10 MB each. Implement a function to allocate blocks sequentially for new photographs based on their sizes. Display the allocation status after each operation (e.g., adding a new photograph, deleting a photograph).

Additional Considerations: Evaluate the efficiency of the sequential allocation strategy in terms of disk space utilization and access times for retrieving files. Consider how fragmentation might affect storage efficiency over time as more files are added and deleted.

```python
class DigitalArchive:
    def __init__(self, total_blocks):
        self.total_blocks = total_blocks
        self.disk_blocks = [False] * total_blocks

    def allocate_blocks(self, file_name, file_size):
        # Write code here

    def delete_file(self, file_name):
        # Write code here

    def calculate_blocks_needed(self, file_size):
        # Write code here

    def display_allocation_status(self):
        # Write code here

# Driver Code
digital_archive = DigitalArchive (100)

digital_archive.allocate_blocks ("Portrait1.jpg", 15)
digital_archive.allocate_blocks ("Landscape1.jpg", 10)
digital_archive.allocate_blocks ("Architecture1.jpg", 7)

digital_archive.display_allocation_status()

digital_archive.delete_file("Landscape1.jpg")

print("\nAfter deleting Landscape1.jpg:")
digital_archive.display_allocation_status()
```

3.5 EnterpriseX

You are a systems engineer working for a large multinational corporation that specializes in developing operating systems for enterprise-level servers. Your team is tasked with designing a file allocation strategy that optimizes data access and storage efficiency for the company's new flagship operating system, OS EnterpriseX.

OS EnterpriseX is designed to handle large-scale data processing and storage for corporate clients. One of the critical requirements is to efficiently manage and access large files, such as databases and multimedia content, stored on disk drives. The system needs to ensure quick access to specific data blocks while maintaining efficient storage utilization.

Requirements:
File Types: Files managed by OS EnterpriseX include database files (structured data) and multimedia files (e.g., videos, images).
Storage System: The operating system uses a disk storage system divided into blocks of fixed size (e.g., 8 KB per block).
Index Sequential File Allocation Strategy: This strategy involves maintaining an index (typically a B-tree or a hash table) that allows direct access to data blocks associated with each file. The index helps in quickly locating the starting block of a file and its subsequent blocks, providing efficient data retrieval.
File Allocation Strategy: Index Sequential File Allocation
Index Management: Maintain an index structure that maps each file to its corresponding disk blocks. The index allows for direct access to specific blocks using file identifiers or keys.
Sequential Access: Files are stored sequentially in terms of logical organization, but physical storage is optimized using the index structure.
Tasks to Implement:

Design a program or algorithm that simulates the index sequential file allocation strategy for managing and accessing files in OS EnterpriseX. Implement functions to allocate blocks for new files, manage the index structure, retrieve data blocks based on file identifiers, and display the current allocation status.

Evaluate the efficiency of the index sequential strategy in terms of access times and storage utilization compared to other strategies (e.g., purely sequential or direct allocation). Assume the storage system has 1000 blocks of 8 KB each. Implement functions to allocate blocks using the index sequential strategy based on file sizes and manage the index structure for quick access.

Consider scenarios where files are updated, deleted, or new files are added, and evaluate how the index structure adapts to these operations. Evaluate the impact of fragmentation and disk space utilization with the index sequential strategy.

```python
class OperatingSystem:
    def __init__(self, total_blocks):
        self.total_blocks = total_blocks
        self.disk_blocks = [False] * total_blocks  # False indicates block is free
        self.index = {}  # Index to map file identifiers to disk blocks

    def allocate_blocks(self, file_name, file_size):
        # Write code here

    def delete_file(self, file_name):
        # Write code here

    def find_free_blocks(self, required_blocks):
        # Write code here

    def calculate_blocks_needed(self, file_size):
        # Write code here

    def total_free_blocks(self):
        # Write code here

    def display_allocation_status(self):
        # Write code here

# Driver Code
os = OperatingSystem(1000)
```

```
os.allocate_blocks ("database.db", 300)
os.allocate_blocks ("video.mp4", 600)
os.allocate_blocks ("image.jpg", 150)

os.display_allocation_status()
os.delete_file("video.mp4")

print("\n After deleting video.mp4:")
os.display_allocation_status()
```

4. Memory Management

4.1 Memory Variable Technique (MVT)

You are a system administrator responsible for managing a computer system that employs the Memory Variable Technique (MVT) for memory management. Your system receives requests from multiple users to run different processes, each requiring a specific amount of memory.

Your system has a total of 1000 KB of memory available, divided into fixed-size partitions to accommodate different process sizes. The Memory Variable Technique dynamically allocates partitions to processes based on their size, aiming to minimize internal fragmentation and optimize memory usage.

Memory Partitions: The system has three memory partitions with sizes as follows:
Partition 1: 300 KB
Partition 2: 500 KB
Partition 3: 200 KB

Process Requests: Users submit requests to run processes of varying sizes (in KB).

Memory Allocation: Implement algorithms to allocate memory to processes based on the MVT strategy:
Allocate memory to a process in the smallest partition that can accommodate its size.
Track and display the allocation status after each operation (allocation or deallocation).

Tasks to Implement:
Design a program or algorithm that simulates the Memory Variable Technique (MVT) for memory allocation. Implement functions to allocate memory to processes, manage the allocation status, deallocate memory when processes finish, and display the current memory allocation status. Evaluate the efficiency of the MVT strategy in terms of memory utilization and fragmentation management.

Implement functions to handle memory allocation requests from users based on process sizes and available partitions. Display the memory allocation status after each operation (allocation or deallocation).

Additional Considerations:
Consider scenarios where memory requests exceed available partition sizes and implement appropriate handling (e.g., fragmentation management, rejection of oversized requests).

```
class MemoryVariableTechnique:
    def __init__(self, partitions):
        self.partitions = partitions
        self.memory_map = {1: [False] * partitions[1],
                           2: [False] * partitions[2],
                           3: [False] * partitions[3]}
        self.processes = {}

    def allocate_memory(self, process_id, size):
        # Write code here
    def deallocate_memory(self, process_id):
        # Write code here

    def display_memory_status(self):
        # Write code here

# Driver Code
mvt = MemoryVariableTechnique({1: 300, 2: 500, 3: 200})

mvt.allocate_memory(1, 150)
mvt.allocate_memory(2, 400)
mvt.allocate_memory(3, 100)

mvt.display_memory_status()

mvt.deallocate_memory(2)

print("\nAfter deallocating Process 2:")
mvt.display_memory_status()
```

4.2 Best Fit Memory Allocation

You are a software developer working on an operating system project that requires implementing memory management techniques. One of the critical components is the Best Fit memory allocation algorithm, which aims to allocate memory blocks to processes in a way that minimizes wastage and fragmentation.

Your operating system is designed to manage a system with a total of 800 KB of memory available. Processes of varying sizes are submitted to the system for execution, each requiring a specific amount of memory. Your task is to implement and simulate the Best Fit memory allocation algorithm to efficiently allocate memory blocks to these processes.

Memory Blocks: The system has a total of 8 memory blocks available, each with a fixed size.
Block 1: 100 KB
Block 2: 200 KB
Block 3: 50 KB
Block 4: 150 KB
Block 5: 300 KB
Block 6: 80 KB
Block 7: 120 KB
Block 8: 200 KB
Process Requests: Users submit requests to run processes of varying sizes (in KB).
Best Fit Algorithm: Implement the Best Fit algorithm to allocate the smallest possible block that can accommodate each process size. If no suitable block is found, reject the process request.
Memory Utilization: Track and display the allocation status after each operation (allocation or rejection).
Tasks to Implement:

Design a program or algorithm that simulates the Best Fit memory allocation strategy. Implement functions to allocate memory blocks to processes, manage the allocation status, reject processes when appropriate, and display the current memory allocation status. Evaluate the efficiency of the Best Fit algorithm in terms of memory utilization and fragmentation management.

Implement functions to handle memory allocation requests from users based on process sizes and available memory blocks. Display the memory allocation status after each operation (allocation or rejection).

Additional Considerations:

Consider scenarios where multiple requests are submitted simultaneously and evaluate how the Best Fit algorithm manages memory blocks efficiently. Evaluate the impact of fragmentation and memory utilization with the Best Fit algorithm compared to other allocation strategies.

```python
class BestFitMemoryManager:
    def __init__(self, memory_blocks):
        self.memory_blocks = memory_blocks
        self.available_blocks = {i: block for i, block in enumerate(memory_blocks)}
        self.processes = {}

    def allocate_memory(self, process_id, size):
        # Write code here

    def deallocate_memory(self, process_id):
        # Write code here

    def display_memory_status(self):
        # Write code here

# Driver Code
memory_manager = BestFitMemoryManager([100, 200, 50, 150, 300, 80, 120, 200])

memory_manager.allocate_memory(1, 70)
memory_manager.allocate_memory(2, 180)
memory_manager.allocate_memory(3, 250)

memory_manager.display_memory_status()

memory_manager.deallocate_memory(2)

print("\nAfter deallocating Process 2:")
memory_manager.display_memory_status()
```

4.3 Worst Fit Memory Allocation

You are a system analyst tasked with designing memory management techniques for a new operating system. One of the techniques to implement is the Worst Fit memory allocation algorithm, which prioritizes allocating the largest available memory block to processes.

Your operating system is designed to manage a system with a total of 1200 KB of memory available, divided into fixed-size memory blocks. Processes of varying sizes are submitted to the system for execution, each requiring a specific amount of memory. Your task is to implement and simulate the Worst Fit memory allocation algorithm to efficiently allocate memory blocks to these processes.

Memory Blocks: The system has a total of 10 memory blocks available, each with a fixed size.
Block 1: 150 KB
Block 2: 300 KB
Block 3: 100 KB
Block 4: 200 KB
Block 5: 250 KB
Block 6: 50 KB
Block 7: 350 KB
Block 8: 180 KB
Block 9: 120 KB
Block 10: 200 KB
Process Requests: Users submit requests to run processes of varying sizes (in KB).

Worst Fit Algorithm: Implement the Worst Fit algorithm to allocate the largest possible block that can accommodate each process size. If no suitable block is found, reject the process request.
Memory Utilization: Track and display the allocation status after each operation (allocation or rejection).

Tasks to Implement: Design a program or algorithm that simulates the Worst Fit memory allocation strategy. Implement functions to allocate memory blocks to processes, manage the allocation status, reject processes when appropriate, and display the current memory allocation status. Evaluate the efficiency of the Worst Fit algorithm in terms of memory utilization and fragmentation management.

Implement functions to handle memory allocation requests from users based on process sizes and available memory blocks. Display the memory allocation status after each operation (allocation or rejection).
Additional Considerations:
Consider scenarios where multiple requests are submitted simultaneously and evaluate how the Worst Fit algorithm manages memory blocks efficiently. Evaluate the impact of fragmentation and memory utilization with the Worst Fit algorithm compared to other allocation strategies.

```
class WorstFitMemoryManager:
    def __init__(self, memory_blocks):
        self.memory_blocks = memory_blocks
        self.available_blocks = {i: block for i, block in enumerate(memory_blocks)}
        self.processes = {}

    def allocate_memory(self, process_id, size):
        # Write code here

    def deallocate_memory(self, process_id):
        # Write code here

    def display_memory_status(self):
        # Write code here

# Driver Code
memory_manager = WorstFitMemoryManager([150, 300, 100, 200, 250, 50, 350, 180, 120,
200])

memory_manager.allocate_memory(1, 180)
memory_manager.allocate_memory(2, 400)
memory_manager.allocate_memory(3, 120)

memory_manager.display_memory_status()

memory_manager.deallocate_memory(2)

print("\nAfter deallocating Process 2:")
memory_manager.display_memory_status()
```

4.4 Multiprogramming with a Fixed Number of Tasks (MFT)

You are a systems engineer working on a legacy mainframe system that employs the MFT memory management technique. The system is designed to handle multiple processes simultaneously by dividing memory into fixed-size partitions, each capable of holding a single process.

Your mainframe system has a total of 6400 KB of memory available, divided into fixed-size partitions. Processes of varying sizes are submitted to the system for execution, each requiring a specific amount of memory. Your task is to implement and simulate the MFT memory management technique to efficiently allocate memory partitions to these processes.

Memory Partitions: The system memory is divided into fixed-size partitions to accommodate processes.
Partition Size: 800 KB each.
Total Partitions: 8 partitions in total.
Process Requests: Users submit requests to run processes of varying sizes (in KB).
MFT Algorithm: Implement the MFT algorithm to allocate the smallest available partition that can accommodate each process size. If no suitable partition is found, reject the process request.
Memory Utilization: Track and display the allocation status after each operation (allocation or rejection).
Tasks to Implement:

Design a program or algorithm that simulates the MFT memory management technique. Implement functions to allocate memory partitions to processes, manage the allocation status, reject processes when appropriate, and display the current memory allocation status. Evaluate the efficiency of the MFT technique in terms of memory utilization and management of partitions.

Implement functions to handle memory allocation requests from users based on process sizes and available partitions. Display the memory allocation status after each operation (allocation or rejection).

Additional Considerations:
Consider scenarios where multiple requests are submitted simultaneously and evaluate how the MFT algorithm manages partitions efficiently. Evaluate the impact of fragmentation and memory utilization with the MFT technique compared to other allocation strategies.

```
class MFTMemoryManager:
    def __init__(self, num_partitions, partition_size):
        self.num_partitions = num_partitions
        self.partition_size = partition_size
        self.available_partitions = [True] * num_partitions
        self.processes = {}

    def allocate_memory(self, process_id, size):
        # Write code here

    def deallocate_memory(self, process_id):
        # Write code here

    def display_memory_status(self):
        # Write code here

# Driver Code
memory_manager = MFTMemoryManager(num_partitions=8, partition_size=800)

memory_manager.allocate_memory(1, 600)
memory_manager.allocate_memory(2, 900)
memory_manager.allocate_memory(3, 400)

memory_manager.display_memory_status()

memory_manager.deallocate_memory(2)

print("\nAfter deallocating Process 2:")
memory_manager.display_memory_status()
```

4.5 Simulating Paging Memory Management

You are a software engineer tasked with implementing memory management for a new operating system that utilizes the Paging technique. Paging divides the physical memory into fixed-size blocks (pages) and manages processes by allocating these pages dynamically.

Your operating system has a total of 4000 KB of physical memory available, divided into fixed-size pages. Processes of varying sizes are submitted to the system for execution, each requiring a specific amount of memory. Your task is to implement and simulate the Paging memory management technique to efficiently allocate memory pages to these processes.

Memory Pages: The system memory is divided into fixed-size pages to accommodate processes.

Page Size: 200 KB each.

Total Pages: 20 pages in total.

Process Requests: Users submit requests to run processes of varying sizes (in KB).

Paging Algorithm: Implement the Paging algorithm to allocate the required number of pages to each process dynamically. If no sufficient contiguous pages are available, reject the process request.

Memory Utilization: Track and display the allocation status after each operation (allocation or rejection).

Design a program or algorithm that simulates the Paging memory management technique. Implement functions to allocate memory pages to processes, manage the allocation status, reject processes when appropriate, and display the current memory allocation status. Evaluate the efficiency of the Paging technique in terms of memory utilization and management of pages.

Implement functions to handle memory allocation requests from users based on process sizes and available pages. Display the memory allocation status after each operation (allocation or rejection).

Additional Considerations:

Consider scenarios where multiple requests are submitted simultaneously and evaluate how the Paging algorithm manages pages efficiently. Evaluate the impact of fragmentation and memory utilization with the Paging technique compared to other allocation strategies.

```python
class PagingMemoryManager:
    def __init__(self, num_pages, page_size):
        self.num_pages = num_pages
        self.page_size = page_size
        self.available_pages = [True] * num_pages
        self.processes = {}

    def allocate_memory(self, process_id, size):
        # Write code here

    def deallocate_memory(self, process_id):
        # Write code here

    def display_memory_status(self):
        # Write code here

# Driver Code
memory_manager = PagingMemoryManager(num_pages=20, page_size=200)

memory_manager.allocate_memory(1, 400)
memory_manager.allocate_memory(2, 600)
memory_manager.allocate_memory(3, 300)

memory_manager.display_memory_status()
memory_manager.deallocate_memory(2)

print("\nAfter deallocating Process 2:")
memory_manager.display_memory_status()
```

5. Contiguous Memory Allocation

5.1 Futuristic Space Station

In a futuristic space station, the central computer manages memory allocation using the worst-fit technique. The memory is divided into fixed-size blocks, and whenever a program requests memory allocation, the system searches for the largest available block that can accommodate the program's size. One day, the space station receives requests from three different programs:

Program A requires 150 units of memory.

Program B requires 300 units of memory.

Program C requires 200 units of memory.

The memory blocks available in the system are as follows:

Block 1: 400 units

Block 2: 250 units

Block 3: 350 units

Block 4: 200 units

Block 5: 150 units

Assume the worst-fit algorithm starts its search from the beginning of the memory and scans sequentially to find the largest block that fits each program's requirement.

Task: Determine how the worst-fit algorithm assigns memory blocks to each program, and calculate the remaining memory after all programs are allocated.

Explanation:

Program A (150 units): The worst-fit algorithm will select Block 1 (400 units) since it's the largest block available that can accommodate Program A. After allocating 150 units to Program A, Block 1 will have 250 units remaining.

Program B (300 units): Now, the system will search for the largest block that fits Program B's requirement.

The largest block available is Block 3 (350 units). After allocating 300 units to Program B, Block 3 will have 50 units remaining.

Program C (200 units): Finally, Program C needs a block of 200 units. The largest block available now is Block 2 (250 units). After allocating 200 units to Program C, Block 2 will have 50 units remaining.

Remaining Memory:

Block 1: 250 units (after Program A)

Block 2: 50 units (after Program C)

Block 3: 50 units (after Program B)

Block 4: 200 units

Block 5: 150 units

Thus, after allocating memory to all three programs using the worst-fit algorithm, the space station's computer system has 250 units + 50 units + 50 units = 350 units of memory remaining.

```
def worst_fit_allocation(memory_blocks, program_requests):
    # Write code here

# Driver Code
memory_blocks = [400, 250, 350, 200, 150]

program_requests = {'Program A': 150, 'Program B': 300,'Program C': 200 }

allocations, remaining_memory = worst_fit_allocation(memory_blocks, program_requests)

for program, allocated_memory in allocations.items():
     print(f"{program} allocated {allocated_memory} units of memory.")

print("\n Remaining Memory in Memory Blocks:")
for i in range(len(memory_blocks)):
     print(f"Block {i+1}: {memory_blocks[i]} units")
```

5.2 CentralAI

In a futuristic city where advanced artificial intelligence (AI) systems govern public services, a central AI hub named CentralAI operates using a sophisticated memory management system. CentralAI oversees various subsystems, each requiring different amounts of memory to function efficiently. To optimize its memory usage, CentralAI employs the best-fit contiguous memory allocation technique. CentralAI manages a total of 8000 units of memory. Throughout the day, it receives memory requests from different subsystems in the following sequence:

SubsysA requests 1500 units of memory.

SubsysB requests 1000 units of memory.

SubsysC requests 700 units of memory.

SubsysD requests 2200 units of memory.

SubsysE requests 500 units of memory.

SubsysF requests 1200 units of memory.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Best-Fit Allocation Process: Show step-by-step how CentralAI allocates memory to each subsystem using the best-fit technique. Detail which blocks of memory are allocated to each subsystem and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the best-fit technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the best-fit technique in this scenario. Are there any disadvantages or concerns CentralAI should be aware of?

Explanation: Illustration of Best-Fit Allocation Process:

CentralAI starts with 8000 units of memory available.

SubsysA (1500 units) would be allocated the smallest block (if any is available), leaving 6500 units.

SubsysB (1000 units) would then take the smallest available block that fits (5500 units left).

SubsysC (700 units) would take the smallest available block (4800 units left).

SubsysD (2200 units) would take the smallest available block (2600 units left).

SubsysE (500 units) would take the smallest available block (2100 units left).

SubsysF (1200 units) would take the smallest available block (900 units left).

Detailed steps should show how the best-fit method minimizes wasted memory by allocating the smallest block that fits each subsystem's request.

Memory Utilization:

After all requests are processed, calculate the sum of memory allocated to each subsystem to find the total memory utilized.

Fragmentation Analysis:

Discuss potential fragmentation issues with best-fit, where smaller gaps of memory are left scattered after allocation, potentially making it difficult to allocate larger blocks that may arrive later. Mention strategies such as compaction or dynamic memory management to address fragmentation over time.

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here
    def __str__(self):
        status = "Free" if not self.allocated else f"Allocated to {self.process_name}"
        return f"[Start: {self.start}, Size: {self.size}, Status: {status}]"

class MemoryManager:
    def __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]

    def allocate_memory(self, process_name, size):
        # Write code here

    def print_memory_status(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(8000)

requests = [("SubsysA", 1500), ("SubsysB", 1000), ("SubsysC", 700),("SubsysD", 2200),
        ("SubsysE", 500), ("SubsysF", 1200)]

for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size)
    if allocated:
       memory_manager.print_memory_status()

memory_manager.print_total_memory_used()
```

5.3 MetroCentral

In a bustling metropolis where a central command center, MetroCentral, manages all public transportation systems using advanced computer systems, efficient memory management is critical. MetroCentral employs the first-fit contiguous memory allocation technique to handle memory requests from various subsystems that control different aspects of the transportation network. MetroCentral operates with a total of 5000 units of memory. Throughout the day, it receives memory requests from different subsystems in the following sequence:

TrafficControl requests 1200 units of memory.

RoutePlanning requests 800 units of memory.

VehicleMonitoring requests 1500 units of memory.

PassengerInformation requests 600 units of memory.

MaintenanceLogistics requests 700 units of memory.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the First-Fit Allocation Process: Show step-by-step how MetroCentral allocates memory to each subsystem using the first-fit technique. Detail which blocks of memory are allocated to each subsystem and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the first-fit technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the first-fit technique in this scenario. Are there any disadvantages or concerns MetroCentral should be aware of?

Explanation:

Illustration of First-Fit Allocation Process:

MetroCentral starts with 5000 units of memory available.

TrafficControl (1200 units) would be allocated the first available block that fits (entire memory block of 5000 units used).

RoutePlanning (800 units) would be allocated the next available block that fits (4000 units left).

VehicleMonitoring (1500 units) would be allocated the next available block that fits (2500 units left).

PassengerInformation (600 units) would be allocated the next available block that fits (1900 units left).

MaintenanceLogistics (700 units) would be allocated the next available block that fits (1200 units left).

Detailed steps should show how the first-fit method allocates memory based on the first available block that meets or exceeds the requested size.

Memory Utilization:

After all requests are processed, calculate the sum of memory allocated to each subsystem to find the total memory utilized.

Fragmentation Analysis:

Discuss potential fragmentation issues with first-fit, where smaller gaps of memory might be left unused between allocated blocks, potentially leading to inefficient memory usage over time.

Mention strategies such as compaction or dynamic memory management to address fragmentation and optimize memory utilization.

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here
class MemoryManager:
    def __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]

    def allocate_memory(self, process_name, size):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(5000)

requests = [
        ("TrafficControl", 1200),
        ("RoutePlanning", 800),
        ("VehicleMonitoring", 1500),
        ("PassengerInformation", 600),
        ("MaintenanceLogistics", 700)
        ]

for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size)
    if allocated:
       memory_manager.print_memory_status()

memory_manager.print_total_memory_used()
```

5.4 AI Lab

In a research institute focused on artificial intelligence (AI) advancements, the AI Lab manages its computational resources using a first-fit contiguous memory allocation technique. The AI Lab is responsible for running various experiments and simulations, each requiring different amounts of memory. The AI Lab operates with a total of 6000 units of memory. Throughout the day, it receives memory requests from different research projects in the following sequence:

Project A requests 1500 units of memory.

Project B requests 1000 units of memory.

Project C requests 700 units of memory.

Project D requests 2200 units of memory.

Project E requests 500 units of memory.

Project F requests 1200 units of memory.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the First-Fit Allocation Process: Show step-by-step how the AI Lab allocates memory to each research project using the first-fit technique. Detail which blocks of memory are allocated to each project and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the first-fit technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the first-fit technique in this scenario. Are there any disadvantages or concerns the AI Lab should be aware of?

Explanation: The AI Lab starts with 6000 units of memory available.

Project A (1500 units) would be allocated the first available block that fits (entire memory block of 6000 units used).

Project B (1000 units) would be allocated the next available block that fits (4500 units left).

Project C (700 units) would be allocated the next available block that fits (3800 units left).

Project D (2200 units) would be allocated the next available block that fits (1600 units left).

Project E (500 units) would be allocated the next available block that fits (1100 units left).

Project F (1200 units) would be allocated the next available block that fits (0 units left).

Detailed steps should show how the first-fit method allocates memory based on the first available block that meets or exceeds the requested size.

Memory Utilization:

After all requests are processed, calculate the sum of memory allocated to each project to find the total memory utilized.

Fragmentation Analysis:

Discuss potential fragmentation issues with first-fit, where smaller gaps of memory might be left unused between allocated blocks, potentially leading to inefficient memory usage over time. Mention strategies such as compaction or dynamic memory management to address fragmentation and optimize memory utilization.

```
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here
    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, total_memory):
        # Write code here

    def allocate_memory(self, process_name, size):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(6000)

requests = [
        ("Project A", 1500),
        ("Project B", 1000),
        ("Project C", 700),
        ("Project D", 2200),
        ("Project E", 500),
        ("Project F", 1200)
          ]

for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size)
    if allocated:
       memory_manager.print_memory_status()

memory_manager.print_total_memory_used()
```

5.5 MedTech Hospital

In a bustling city where a central hospital, MedTech Hospital, manages its patient records and medical data using sophisticated computer systems, efficient memory management is crucial. MedTech Hospital employs the best-fit contiguous memory allocation technique to optimize its use of memory resources. MedTech Hospital operates with a total of 8000 units of memory. Throughout a busy day, it receives memory requests from different departments and systems in the following sequence:

Emergency Department requests 2000 units of memory.

Cardiology Department requests 1500 units of memory.

Laboratory Information System requests 1200 units of memory.

Radiology Department requests 1800 units of memory.

Patient Management System requests 1000 units of memory.

Pharmacy System requests 600 units of memory.

Surgical Services requests 2200 units of memory.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Best-Fit Allocation Process: Show step-by-step how MedTech Hospital allocates memory to each department or system using the best-fit technique. Detail which blocks of memory are allocated to each request and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the best-fit technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the best-fit technique in this scenario. Are there any disadvantages or concerns MedTech Hospital should be aware of?

Explanation: MedTech Hospital starts with 8000 units of memory available.

Emergency Department (2000 units) would be allocated the smallest available block that fits (entire memory block of 8000 units used).

Cardiology Department (1500 units) would be allocated the smallest available block that fits (6500 units left).

Laboratory Information System (1200 units) would be allocated the smallest available block that fits (5300 units left).

Radiology Department (1800 units) would be allocated the smallest available block that fits (3500 units left).

Patient Management System (1000 units) would be allocated the smallest available block that fits (2500 units left).

Pharmacy System (600 units) would be allocated the smallest available block that fits (1900 units left).

Surgical Services (2200 units) would be allocated the smallest available block that fits (0 units left).

Detailed steps should show how the best-fit method allocates memory based on the smallest available block that meets or exceeds the requested size.

Memory Utilization:

After all requests are processed, calculate the sum of memory allocated to each department or system to find the total memory utilized.

Fragmentation Analysis:

Discuss potential fragmentation issues with best-fit, where smaller gaps of memory might be left unused between allocated blocks, potentially leading to inefficient memory usage over time. Mention strategies such as compaction or dynamic memory management to address fragmentation and optimize memory utilization.

```python
class MemoryBlock:
    def __init__(self, start, size):
        self.start = start
        self.size = size
        self.allocated = False
        self.process_name = None

    def is_free(self):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, total_memory):
        self.total_memory = total_memory
        self.memory_blocks = [MemoryBlock(0, total_memory)]

    def allocate_memory(self, process_name, size):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(8000)

requests = [
        ("Emergency Department", 2000),
        ("Cardiology Department", 1500),
        ("Laboratory Information System", 1200),
        ("Radiology Department", 1800),
        ("Patient Management System", 1000),
        ("Pharmacy System", 600),
        ("Surgical Services", 2200)
          ]

for request in requests:
    process_name, size = request
    allocated = memory_manager.allocate_memory(process_name, size)
    if allocated:
       memory_manager.print_memory_status()

memory_manager.print_total_memory_used()
```

6. Paging Memory Management

6.1 CloudTech

In a fast-growing tech startup, CloudTech Inc., which specializes in cloud computing services, efficient memory management is critical to ensure optimal performance and resource utilization. CloudTech Inc. employs the paging technique to manage memory across its cloud servers. CloudTech Inc. operates multiple cloud servers, each equipped with a paging system. The main memory of each server is divided into fixed-size pages, and processes running on these servers request memory in terms of these pages. Each page can hold a fixed amount of data.

Each cloud server has a total of 100 pages of main memory. Each page can hold up to 4 units of data. Processes running on the servers request memory in terms of the number of pages they need. Throughout a busy day, several processes make memory requests in the following sequence:

Process A requests 25 pages.

Process B requests 15 pages.

Process C requests 30 pages.

Process D requests 12 pages.

Process E requests 20 pages.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Paging Memory Management Process: Show step-by-step how CloudTech Inc. allocates memory to each process using the paging technique. Detail which pages are allocated to each process and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the paging technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the paging technique in this scenario. Are there any disadvantages or concerns CloudTech Inc. should be aware of?

Explanation: Each cloud server starts with 100 pages of main memory.

Process A (25 pages) would be allocated consecutive pages starting from the beginning (pages 0 to 24).

Process B (15 pages) would be allocated the next available consecutive pages (pages 25 to 39).

Process C (30 pages) would be allocated consecutive pages (pages 40 to 69).

Process D (12 pages) would be allocated consecutive pages (pages 70 to 81).

Process E (20 pages) would be allocated consecutive pages (pages 82 to 101).

Detailed steps should show how the paging technique allocates memory based on consecutive pages for each process request.

Memory Utilization:

After all requests are processed, calculate the total number of pages allocated to processes to find the total memory utilized.

```
class Page:
    def __init__(self, page_id, process_name=None):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, num_pages, page_size):
        self.num_pages = num_pages
        self.page_size = page_size
        self.pages = [Page(page_id) for page_id in range(num_pages)]

    def allocate_memory(self, process_name, num_pages_requested):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(num_pages=100, page_size=4)

requests = [
        ("Process A", 25),
        ("Process B", 15),
        ("Process C", 30),
        ("Process D", 12),
        ("Process E", 20)
          ]

for request in requests:
    process_name, num_pages_requested = request
    allocated = memory_manager.allocate_memory(process_name, num_pages_requested)
    if allocated:
       print(f"Allocated {num_pages_requested} pages for {process_name}")
       memory_manager.print_memory_status()
    else:
       print(f"Not enough memory available for {process_name} (requested
{num_pages_requested} pages)")

memory_manager.print_total_memory_used()
```

6.2 EduTech University

In a large educational institute, EduTech University, which handles vast amounts of student and administrative data, efficient memory management is crucial for maintaining smooth operations of its IT infrastructure. EduTech University utilizes the paging technique to manage memory across its various departments and systems. EduTech University operates a centralized server system with a total of 200 pages of main memory. Each page can accommodate up to 8 units of data. Throughout a typical academic day, different departments and systems within the university make memory requests in the following sequence:

Student Records System requests 40 pages.

Faculty Management System requests 25 pages.

Library Information System requests 30 pages.

Online Learning Platform requests 35 pages.

Research Database requests 50 pages.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Paging Memory Management Process: Show step-by-step how EduTech University allocates memory to each department or system using the paging technique. Detail which pages are allocated to each request and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the paging technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the paging technique in this scenario. Are there any disadvantages or concerns EduTech University should be aware of?

Explanation: EduTech University starts with 200 pages of main memory.

Student Records System (40 pages) would be allocated consecutive pages starting from the beginning (pages 0 to 39).

Faculty Management System (25 pages) would be allocated the next available consecutive pages (pages 40 to 64).

Library Information System (30 pages) would be allocated consecutive pages (pages 65 to 94).

Online Learning Platform (35 pages) would be allocated consecutive pages (pages 95 to 129).

Research Database (50 pages) would be allocated consecutive pages (pages 130 to 179).

Detailed steps should show how the paging technique allocates memory based on consecutive pages for each department or system request.

Memory Utilization:

After all requests are processed, calculate the total number of pages allocated to each department or system to find the total memory utilized.

```python
class Page:
    def __init__(self, page_id, process_name=None):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, num_pages, page_size):
        # Write code here

    def allocate_memory(self, process_name, num_pages_requested):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(num_pages=200, page_size=8)

requests = [
        ("Student Records System", 40),
        ("Faculty Management System", 25),
        ("Library Information System", 30),
        ("Online Learning Platform", 35),
        ("Research Database", 50)
          ]

for request in requests:
    process_name, num_pages_requested = request
    allocated = memory_manager.allocate_memory(process_name, num_pages_requested)
    if allocated:
       print(f"Allocated {num_pages_requested} pages for {process_name}")
       memory_manager.print_memory_status()
    else:
       print(f"Not enough memory available for {process_name} (requested
{num_pages_requested} pages)")
memory_manager.print_total_memory_used()
```

6.3 GameTech Studios

In a bustling gaming company, GameTech Studios, which hosts multiplayer online games, efficient memory management is crucial to ensure seamless gameplay experiences for millions of players worldwide. GameTech Studios employs the paging technique to manage memory across its gaming servers. GameTech Studios operates multiple gaming servers, each equipped with a paging system. The main memory of each server is divided into fixed-size pages, and game sessions running on these servers request memory in terms of these pages. Each page can store a fixed amount of game data.

Each gaming server has a total of 256 pages of main memory.

Each page can hold up to 16 units of game data.

Throughout a peak gaming period, several game sessions make memory requests in the following sequence:

Game Session 1 requests 32 pages.

Game Session 2 requests 20 pages.

Game Session 3 requests 40 pages.

Game Session 4 requests 18 pages.

Game Session 5 requests 25 pages.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Paging Memory Management Process: Show step-by-step how GameTech Studios allocates memory to each game session using the paging technique. Detail which pages are allocated to each request and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the paging technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the paging technique in this scenario. Are there any disadvantages or concerns GameTech Studios should be aware of?

Explanation:

Each gaming server starts with 256 pages of main memory.

Game Session 1 (32 pages) would be allocated consecutive pages starting from the beginning (pages 0 to 31).

Game Session 2 (20 pages) would be allocated the next available consecutive pages (pages 32 to 51).

Game Session 3 (40 pages) would be allocated consecutive pages (pages 52 to 91).

Game Session 4 (18 pages) would be allocated consecutive pages (pages 92 to 109).

Game Session 5 (25 pages) would be allocated consecutive pages (pages 110 to 134).

Detailed steps should show how the paging technique allocates memory based on consecutive pages for each game session request.

Memory Utilization:

After all requests are processed, calculate the total number of pages allocated to each game session to find the total memory utilized.

```
class Page:
    def __init__(self, page_id, process_name=None):
        # Write code here

    def allocate(self, process_name):
        # Write code here
    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, num_pages, page_size):
        # Write code here

    def allocate_memory(self, process_name, num_pages_requested):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(num_pages=256, page_size=16)

requests = [
        ("Game Session 1", 32),
        ("Game Session 2", 20),
        ("Game Session 3", 40),
        ("Game Session 4", 18),
        ("Game Session 5", 25)
           ]

for request in requests:
    process_name, num_pages_requested = request
    allocated = memory_manager.allocate_memory(process_name, num_pages_requested)
    if allocated:
       print(f"Allocated {num_pages_requested} pages for {process_name}")
       memory_manager.print_memory_status()
    else:
       print(f"Not enough memory available for {process_name} (requested
{num_pages_requested} pages)")

memory_manager.print_total_memory_used()
```

6.4 MedCare Hospital

In a large hospital, MedCare Hospital, which manages extensive patient records and medical data, efficient memory management is critical to ensure timely access to patient information and smooth operation of medical services. MedCare Hospital utilizes the paging technique to manage memory across its various departments and systems. MedCare Hospital operates a centralized IT system with a total of 300 pages of main memory. Each page can store a fixed amount of patient data. Throughout a busy day, different departments and systems within the hospital make memory requests in the following sequence:

Emergency Department requests 50 pages.

Radiology Department requests 35 pages.

Laboratory Department requests 45 pages.

Patient Records System requests 60 pages.

Surgery Department requests 40 pages.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Paging Memory Management Process: Show step-by-step how MedCare Hospital allocates memory to each department or system using the paging technique. Detail which pages are allocated to each request and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the paging technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the paging technique in this scenario. Are there any disadvantages or concerns MedCare Hospital should be aware of?

Explanation:

MedCare Hospital starts with 300 pages of main memory.

Emergency Department (50 pages) would be allocated consecutive pages starting from the beginning (pages 0 to 49).

Radiology Department (35 pages) would be allocated the next available consecutive pages (pages 50 to 84).

Laboratory Department (45 pages) would be allocated consecutive pages (pages 85 to 129).

Patient Records System (60 pages) would be allocated consecutive pages (pages 130 to 189).

Surgery Department (40 pages) would be allocated consecutive pages (pages 190 to 229).

Detailed steps should show how the paging technique allocates memory based on consecutive pages for each department or system request.

Memory Utilization:

After all requests are processed, calculate the total number of pages allocated to each department or system to find the total memory utilized.

```python
class Page:
    def __init__(self, page_id, process_name=None):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, num_pages, page_size):
        # Write code here

    def allocate_memory(self, process_name, num_pages_requested):
        # Write code here

    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(num_pages=300, page_size=1)
```

```
requests = [
        ("Emergency Department", 50),
        ("Radiology Department", 35),
        ("Laboratory Department", 45),
        ("Patient Records System", 60),
        ("Surgery Department", 40)
          ]

for request in requests:
    process_name, num_pages_requested = request
    allocated = memory_manager.allocate_memory(process_name, num_pages_requested)
    if allocated:
       print(f"Allocated {num_pages_requested} pages for {process_name}")
       memory_manager.print_memory_status()
    else:
       print(f"Not enough memory available for {process_name} (requested
{num_pages_requested} pages)")

memory_manager.print_total_memory_used()
```

6.5 ShopMart

In a bustling online shopping platform, ShopMart, which manages a vast inventory of products and customer transactions, efficient memory management is crucial to ensure smooth operations and timely access to product data. ShopMart employs the paging technique to manage memory across its servers. ShopMart operates multiple servers to handle customer orders and manage product information. Each server has a total of 512 pages of main memory, and each page can store a fixed amount of product data. Throughout a busy day of operations, different parts of ShopMart's system make memory requests in the following sequence:

Product Catalog Management requests 80 pages.

Order Processing System requests 120 pages.

Customer Database requests 150 pages.

Payment Processing Gateway requests 100 pages.

Inventory Tracking System requests 60 pages.

Assume that the memory is initially empty, and all requests arrive sequentially without any memory being released in between.

Illustrate the Paging Memory Management Process: Show step-by-step how ShopMart allocates memory to each system component using the paging technique. Detail which pages are allocated to each request and explain the decision-making process for each allocation.

Memory Utilization: Calculate and report the total amount of memory utilized after all requests have been processed using the paging technique.

Fragmentation Analysis: Discuss any potential fragmentation issues that might arise with the paging technique in this scenario. Are there any disadvantages or concerns ShopMart should be aware of?

Explanation:

ShopMart starts with 512 pages of main memory per server.

Product Catalog Management (80 pages) would be allocated consecutive pages starting from the beginning (pages 0 to 79).

Order Processing System (120 pages) would be allocated the next available consecutive pages (pages 80 to 199).

Customer Database (150 pages) would be allocated consecutive pages (pages 200 to 349).

Payment Processing Gateway (100 pages) would be allocated consecutive pages (pages 350 to 449).

Inventory Tracking System (60 pages) would be allocated consecutive pages (pages 450 to 509).

Detailed steps should show how the paging technique allocates memory based on consecutive pages for each system component request.

Memory Utilization:

After all requests are processed, calculate the total number of pages allocated to each system component to find the total memory utilized.

```
class Page:
    def __init__(self, page_id, process_name=None):
        # Write code here

    def allocate(self, process_name):
        # Write code here

    def deallocate(self):
        # Write code here

    def __str__(self):
        # Write code here

class MemoryManager:
    def __init__(self, num_pages, page_size):
        # Write code here

    def allocate_memory(self, process_name, num_pages_requested):
        # Write code here
    def print_memory_status(self):
        # Write code here

    def print_total_memory_used(self):
        # Write code here

# Driver Code
memory_manager = MemoryManager(num_pages=512, page_size=1)

requests = [
        ("Product Catalog Management", 80),
        ("Order Processing System", 120),
        ("Customer Database", 150),
        ("Payment Processing Gateway", 100),
        ("Inventory Tracking System", 60)
           ]

for request in requests:
    process_name, num_pages_requested = request
    allocated = memory_manager.allocate_memory(process_name, num_pages_requested)
    if allocated:
       print(f"Allocated {num_pages_requested} pages for {process_name}")
       memory_manager.print_memory_status()
    else:
       print(f"Not enough memory available for {process_name} (requested
{num_pages_requested} pages)")

memory_manager.print_total_memory_used()
```

7. Resource Allocation

7.1 UrbanOS - Resource Allocation Graph (RAG)

In a bustling city, UrbanOS, which operates a complex multitasking operating system, efficient resource management is crucial to ensure smooth operation of various processes and applications. UrbanOS employs a Resource Allocation Graph (RAG) to manage resource allocation and avoid deadlock situations. UrbanOS manages resources such as CPU time, memory, and peripherals across multiple processes running concurrently. Each process may request and release different resources dynamically throughout its execution.

Process Management: UrbanOS supports several processes including:

Process A: Handles user interface interactions and graphics rendering.

Process B: Manages database transactions and file operations.

Process C: Performs complex calculations and simulations.

Process D: Controls network communications and data transfers.

Resource Requests:

Each process in UrbanOS can request multiple resources such as CPU time, memory blocks, and access to peripherals like printers or network devices. Processes dynamically request resources based on their current tasks and release them when no longer needed.

Resource Allocation Graph (RAG):

UrbanOS maintains a Resource Allocation Graph (RAG) to track which processes are currently allocated which resources and which processes are waiting for which resources. The RAG helps UrbanOS detect potential deadlock scenarios where processes are waiting indefinitely for resources held by others.

Illustrate the Resource Allocation Graph (RAG): Draw a diagram or outline showing how UrbanOS constructs and maintains the Resource Allocation Graph (RAG) for the processes described (Process A to Process D). Detail the allocation and request edges between processes and resources.

Deadlock Prevention: Explain how UrbanOS uses the Resource Allocation Graph (RAG) to prevent deadlock situations among processes. Provide examples of deadlock scenarios and how the RAG helps identify and resolve them.

Explanation:

Process A: Requests CPU time and memory blocks for graphics rendering.

Process B: Requests CPU time and file system access for database transactions.

Process C: Requests CPU time and memory blocks for simulations.

Process D: Requests network access and CPU time for network communications.

Draw edges showing which processes are currently allocated which resources and which processes are waiting for which resources (e.g., arrows from processes to resources indicating requests).

Deadlock Prevention:

Example Scenario: Process A holds CPU time and is waiting for memory blocks held by Process C. Process C holds memory blocks and is waiting for CPU time held by Process B. Process B, in turn, is waiting for file system access held by Process D, which is waiting for network access.

RAG Utilization: UrbanOS uses cycle detection in the RAG to identify deadlock situations. If a cycle exists (e.g., Process A -> Process C -> Process B -> Process D -> Process A), UrbanOS can preemptively break one or more edges to resolve the deadlock and allow processes to proceed.

Dynamic Resource Management:

Challenges: Balancing resource utilization across multiple processes, ensuring fair access, and avoiding resource starvation.

Strategies: UrbanOS employs algorithms like Banker's algorithm or deadlock detection algorithms based on the RAG to dynamically allocate and manage resources. It ensures that processes only request resources they can feasibly utilize without causing deadlock or starvation.

```python
class Process:
    def __init__(self, pid, max_resources):
        # Write code here

    def request_resource(self, resource, amount):
        # Write code here

    def release_resource(self, resource, amount):
        # Write code here

    def __str__(self):
        # Write code here

class ResourceManager:
    def __init__(self, resources):
        # Write code here

    def add_process(self, process):
        # Write code here

    def request_resource(self, process_id, resource, amount):
        # Write code here

    def release_resource(self, process_id, resource, amount):
        # Write code here

    def print_processes_status(self):
        # Write code here

# Driver Code
resources = {"CPU": 1, "Memory": 1024}

process1 = Process(1, {"CPU": 1, "Memory": 512})
process2 = Process(2, {"CPU": 1, "Memory": 768})

resource_manager = ResourceManager(resources)
resource_manager.add_process(process1)
resource_manager.add_process(process2)

print("Initial State:")
resource_manager.print_processes_status()

if resource_manager.request_resource(1, "CPU", 1):
   print("Process 1 allocated CPU")
else:
   print("Process 1 failed to allocate CPU")

if resource_manager.request_resource(1, "Memory", 512):
   print("Process 1 allocated Memory")
else:
   print("Process 1 failed to allocate Memory")

resource_manager.print_processes_status()
```

```
if resource_manager.request_resource(2, "Memory", 768):
   print("Process 2 allocated Memory")
else:
   print("Process 2 failed to allocate Memory")

resource_manager.print_processes_status()

if resource_manager.release_resource(1, "Memory", 512):
   print("Process 1 released Memory")
else:
   print("Process 1 failed to release Memory")

resource_manager.print_processes_status()
```

7.2 UrbanOS - Wait-for-Graph (WFG)

UrbanOS, a sophisticated operating system powering a smart city, relies on efficient process management and dependency tracking to ensure seamless operations across its diverse infrastructure. One of the key mechanisms UrbanOS utilizes is the Wait-for-Graph (WFG), which helps manage dependencies and prevent potential deadlock situations among concurrent processes.

Process and Task Management:

UrbanOS oversees a myriad of critical processes that operate simultaneously to support city functions:

Process A: Manages real-time traffic signal adjustments at intersections.

Process B: Handles data aggregation and analytics for environmental sensors.

Process C: Coordinates emergency response dispatch and resource allocation.

Process D: Facilitates centralized billing and utility management.

Concurrency and Dependency Dynamics:

Each process in UrbanOS may depend on resources such as CPU time, memory, network access, and access to shared databases or sensors. Processes dynamically interact, requesting resources and potentially waiting for others to release resources before proceeding.

Wait-for-Graph (WFG) Implementation:

UrbanOS constructs a Wait-for-Graph (WFG) to visually represent dependencies among processes:

Nodes in the graph represent processes. Directed edges between nodes depict dependencies where one process is waiting for another to release a resource. The WFG aids UrbanOS in detecting cyclic dependencies and managing resource contention effectively.

Visualization of Wait-for-Graph (WFG): Illustrate how UrbanOS constructs and updates the Wait-for-Graph (WFG) for the processes described (Process A to Process D). Provide a detailed diagram or outline showing process nodes, dependency edges, and current dependencies.

Deadlock Prevention Strategies: Explain how UrbanOS uses the Wait-for-Graph (WFG) to prevent deadlock situations among processes. Provide examples of potential deadlock scenarios and demonstrate how the WFG helps identify and resolve these scenarios preemptively.

Real-Time Dependency Management: Discuss the challenges UrbanOS faces in managing dependencies and resource contention among concurrent processes. How does the WFG assist in optimizing process scheduling, ensuring timely execution, and maintaining system stability?

Explanation:

Visualization of Wait-for-Graph (WFG):

Process A: Requests CPU time and access to traffic data.

Process B: Requests memory and network access for sensor data processing.

Process C: Requests CPU time and emergency service resources.

Process D: Requests database access and CPU time for utility management.

Draw directed edges in the WFG showing dependencies where one process is waiting for another to release resources, reflecting current dependencies and potential conflicts.

Deadlock Prevention Strategies:

Example Scenario: Process A is waiting for access to traffic data held by Process B. Process B is waiting for CPU time held by Process C, which is waiting for emergency service resources held by Process D, forming a cyclic dependency.

WFG Utilization: UrbanOS employs cycle detection algorithms in the WFG to identify and break potential deadlock cycles. By preemptively releasing and reallocating resources or adjusting scheduling priorities, UrbanOS resolves deadlock scenarios proactively.

Real-Time Dependency Management:

Challenges: Balancing resource demands across processes, minimizing waiting times, and ensuring fair access to critical resources.

Strategies: UrbanOS uses the WFG to monitor and optimize process dependencies in real-time. It implements dynamic scheduling algorithms based on WFG insights to prioritize critical tasks, mitigate resource contention, and enhance overall system responsiveness and reliability.

```python
class Process:
    def __init__(self, pid):
        # Write code here

    def add_dependency(self, process):
        # Write code here

    def remove_dependency(self, process):
        # Write code here

    def __str__(self):
        # Write code here

class WaitForGraph:
    def __init__(self):
        self.processes = {}

    def add_process(self, process):
        # Write code here

    def add_dependency(self, process_pid, dependency_pid):
        # Write code here

    def remove_dependency(self, process_pid, dependency_pid):
        # Write code here

    def detect_deadlocks(self):
        # Write code here

        def detect_cycle(process):
            # Write code here
        return False

    def print_wfg(self):
        # Write code here
# Driver Code
process1 = Process(1)
process2 = Process(2)
process3 = Process(3)
process4 = Process(4)

wfg = WaitForGraph()

wfg.add_process(process1)
wfg.add_process(process2)
wfg.add_process(process3)
wfg.add_process(process4)

wfg.add_dependency(1, 2)
wfg.add_dependency(2, 3)
wfg.add_dependency(3, 4)
wfg.add_dependency(4, 1)

wfg.print_wfg()

if wfg.detect_deadlocks():
   print("Deadlock detected in the system.")
   wfg.remove_dependency(4, 1)
   print("Deadlock resolved.")
else:
   print("No deadlock detected.")

wfg.print_wfg()
```

7.3 The Library Conference

Imagine a conference being held at a university library, where several research teams are working on their projects. The library has a set of resources:

3 computers (R1, R2, R3)

2 projectors (P1, P2)

Each team needs access to both a computer and a projector to complete their work. The teams are as follows:

Team A: Needs 1 computer and 1 projector.

Team B: Needs 1 computer and 1 projector.

Team C: Needs 1 computer and 1 projector.

At any given time, each team is holding a resource while waiting for another resource to complete their work. The following events occur:

Team A is using computer R1 and waiting for projector P1.

Team B is using projector P2 and waiting for computer R2.

Team C is using computer R3 and waiting for projector P2.

Determine if there is a deadlock in the system. If so, describe the circular wait condition and suggest a way to prevent or resolve the deadlock.

Deadlock Detection: To detect deadlock, we need to analyze the resource allocation and the wait conditions.

Team A: Holding R1, waiting for P1.

Team B: Holding P2, waiting for R2.

Team C: Holding R3, waiting for P2.

If we construct a resource allocation graph:

Team A → P1 (waiting for P1)

Team B → R2 (waiting for R2)

Team C → P2 (waiting for P2)

We notice that:

Team A is waiting for P1, which is held by Team B.

Team B is waiting for R2, which is held by Team C.

Team C is waiting for P2, which is held by Team B.

This forms a circular wait condition: Team A → Team B → Team C → Team B

Thus, a deadlock is present.

Resolution Strategy:

Prevention: Implement policies to avoid circular wait. For example, enforce a rule where each team must request all required resources at once.

Avoidance: Use a dynamic approach to resource allocation, such as the Banker's Algorithm, to ensure resources are allocated only if it does not lead to a potential deadlock.

Detection and Recovery: If deadlock detection is used, identify the deadlocked teams and preempt resources by forcibly reallocating them to break the circular wait.

```
import threading
import time

resources = {'computers': ['R1', 'R2', 'R3'], 'projectors': ['P1', 'P2']}

team_requests = {
    'TeamA': {'computer': 'R1', 'projector': 'P1'},
    'TeamB': {'computer': 'R2', 'projector': 'P2'},
    'TeamC': {'computer': 'R3', 'projector': 'P2'}}

allocated_resources = {
    'TeamA': {'computer': None, 'projector': None},
    'TeamB': {'computer': None, 'projector': None},
    'TeamC': {'computer': None, 'projector': None}}

resource_locks = {
    'computers': {res: threading.Lock() for res in resources['computers']},
    'projectors': {res: threading.Lock() for res in resources['projectors']} }

def request_resource(team, resource_type, resource):
    # Write code here

def release_resource(team, resource_type, resource):
    # Write code here

def team_work(team):
    # Write code here

threads = []
for team in team_requests.keys():
    thread = threading.Thread(target=team_work, args=(team,))
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

print("Simulation complete.")
```

7.4 The Dining Philosophers

In a dining hall, there are 5 philosophers sitting around a round table. Each philosopher needs two utensils to eat: one in the left hand and one in the right hand. The table has 5 utensils, each placed between two philosophers.

The philosophers are:

Philosopher 1: Needs utensils U1 and U2.

Philosopher 2: Needs utensils U2 and U3.

Philosopher 3: Needs utensils U3 and U4.

Philosopher 4: Needs utensils U4 and U5.

Philosopher 5: Needs utensils U5 and U1.

Each philosopher picks up the utensil on their left and then tries to pick up the utensil on their right. If they cannot pick up both utensils, they put down the one they have and wait.

Analyze the system for potential deadlock. What strategies can be used to prevent or resolve the deadlock?

Explanation:

Deadlock Detection: To check for deadlock, we observe that:

Philosopher 1 picks up U1 and waits for U2.

Philosopher 2 picks up U2 and waits for U3.

Philosopher 3 picks up U3 and waits for U4.

Philosopher 4 picks up U4 and waits for U5.

Philosopher 5 picks up U5 and waits for U1.

In this scenario, we have a circular wait:

Philosopher 1 → Philosopher 2 → Philosopher 3 → Philosopher 4 → Philosopher 5 → Philosopher 1

Thus, a deadlock is present.

Resolution Strategy:

Prevention: One common approach is to impose an order on the utensils and ensure that philosophers always pick up the lower-numbered utensil first. For example, Philosopher 1 picks up U1 and then U2, Philosopher 2 picks up U2 and then U3, and so on. This approach breaks the circular wait condition.

Avoidance: Use an algorithm like the Banker's Algorithm to ensure that no philosopher will enter a state where they wait indefinitely for resources.

Detection and Recovery: Implement an algorithm to periodically check for deadlocks. If detected, a recovery strategy such as terminating and restarting one or more philosophers can be employed.

```
import threading

class Philosopher(threading.Thread):
    def __init__(self, philosopher_id, left_utensil, right_utensil):
        # Write code here

    def run(self):
        # Write code here

    def think(self):
        # Write code here

    def eat(self):
        # Write code here

# Driver Code
num_philosophers = 5
utensils = [threading.Lock() for _ in range(num_philosophers)]
philosophers = []

for i in range(num_philosophers):
    left_utensil = utensils[i]
    right_utensil = utensils[(i + 1) % num_philosophers]
    philosopher = Philosopher(i + 1, left_utensil, right_utensil)
    philosophers.append(philosopher)

for philosopher in philosophers:
    philosopher.start()
```

7.5 Banker's Algorithm

The Banker's algorithm is a deadlock avoidance algorithm used in operating systems to manage resource allocation and prevent deadlocks. It ensures that resource requests are granted in a way that avoids the possibility of deadlocks. Let's consider a system with five processes (P1, P2, P3, P4, P5) and three resource types (R1, R2, R3). The available resources and maximum resource requirements for each process are as follows:

Available: R1(3), R2(2), R3(2) P1: R1(1), R2(2), R3(2) P2: R1(2), R2(0), R3(1) P3: R1(1), R2(1), R3(1) P4: R1(2), R2(1), R3(0) P5: R1(0), R2(0), R3(2)

To determine if the system is in a safe state using the Banker's algorithm, we need to simulate resource allocation and check if a safe sequence can be formed. Here are the steps involved:

Initialize data structures:

Available: Set it to the available resources.

Allocation: Set it to the current allocation of resources to processes.

Maximum: Set it to the maximum resource requirements of each process.

Need: Calculate it as Maximum - Allocation for each process.

Define a work array and a finish array. Initially, set all elements of the finish array to false.

Find a process P_i such that:

- Finish[i] is false.
- Need[i] <= Available.

If such a process is found, allocate its resources:

- Available = Available + Allocation[i]
- Set Finish[i] to true
- Add Pi to the safe sequence.

Repeat steps 3 and 4 until all processes are either finished or no process can be allocated resources.

If all processes are successfully allocated resources and a safe sequence is formed, then the system is in a safe state. Otherwise, if there is no safe sequence, the system is in an unsafe state, indicating a potential deadlock. In the given example, you can follow the steps of the Banker's algorithm to determine if the system is in a safe state. By simulating resource allocation and checking for a safe sequence, you can assess whether the system can avoid deadlocks.

```python
def is_safe_state(processes, available, allocation, need):
    # Write code here

# Driver Code
processes = ['P1', 'P2', 'P3', 'P4', 'P5']

available = [3, 2, 2]

allocation = [
    [1, 2, 2],
    [2, 0, 1],
    [1, 1, 1],
    [2, 1, 0],
    [0, 0, 2]
]

maximum = [
    [1, 2, 2],
    [2, 0, 1],
    [1, 1, 1],
    [2, 1, 0],
    [0, 0, 2]
          ]

need = [[maximum[i][j] - allocation[i][j] for j in range(len(available))] for i in
range(len(processes))]

is_safe, safe_sequence = is_safe_state(processes, available, allocation, need)

if is_safe:
    print("The system is in a safe state.")
    print("Safe sequence:", safe_sequence)
else:
    print("The system is in an unsafe state. Deadlock may occur.")
```

8. Disk Scheduling

8.1 The Disk Access Dilemma

In a busy computer lab at Techville University, a large disk drive serves the needs of multiple students. The disk drive handles requests from several applications running on different computers. Each request involves accessing a specific track on the disk. The disk drive operates under the First-Come, First-Served (FCFS) scheduling algorithm, meaning that requests are handled in the order they are received, regardless of their location on the disk.

Imagine the disk drive has 1000 tracks, numbered 0 through 999. The disk's current head position is at track 150. Over the course of a few minutes, the following disk access requests are made by different applications:

Request 1: Track 200

Request 2: Track 50

Request 3: Track 800

Request 4: Track 300

Request 5: Track 100

The requests are received in the order listed above.

Problem:

1. Calculate the total head movement required to satisfy all the requests using the FCFS algorithm.
2. Determine the order of tracks that the disk head will visit, starting from the initial position.

Solution:

1. Calculate the Total Head Movement:
    - Initial Head Position: Track 150
    - Order of Requests:
        - Request 1: Track 200
        - Request 2: Track 50
        - Request 3: Track 800
        - Request 4: Track 300
        - Request 5: Track 100
    - Movement Calculation:
        - Move from Track 150 to Track 200: $|200-150|=50$
        - Move from Track 200 to Track 50: $|50-200|=150$
        - Move from Track 50 to Track 800: $|800-50|=750$
        - Move from Track 800 to Track 300: $|300-800|=500$
        - Move from Track 300 to Track 100: $|100-300|=200$
    - Total Head Movement:

      50 + 150 + 750 + 500 + 200 = 1650
2. Order of Tracks Visited:
    - Start at Track 150
    - Move to Track 200 (Request 1)
    - Move to Track 50 (Request 2)
    - Move to Track 800 (Request 3)
    - Move to Track 300 (Request 4)
    - Move to Track 100 (Request 5)

   Order of Tracks Visited: $150 \rightarrow 200 \rightarrow 50 \rightarrow 800 \rightarrow 300 \rightarrow 100$

```
def fcfs_disk_scheduling(initial_head, requests):
    # Write code here

# Driver Code
initial_head_position = 150
disk_requests = [200, 50, 800, 300, 100]
```

```
total_head_movement, tracks_visited = fcfs_disk_scheduling(initial_head_position,
disk_requests)

print("Total Head Movement:", total_head_movement)
print("Order of Tracks Visited:", tracks_visited)
```

8.2 The SSTF Disk Scheduling Challenge

The goal of SSTF is to reduce the total seek time by always selecting the request closest to the current head position. This approach minimizes the movement of the disk head between requests.

Process:

1. **Initial State:** Start with the initial head position (750 in this case).
2. **Select the Closest Request:** At each step, identify the request that is closest to the current head position.
3. **Move to the Closest Request:** Move the disk head to this closest request and mark it as processed.
4. **Update Position and Repeat:** Update the current head position to the new track and repeat the process until all requests are serviced.

Step-by-Step Simulation:
Initial Position: Track 750
Requests: [1200, 500, 900, 1500, 300]
Execution: Current Head: 750
Distances to Requests:
To Track 1200: |1200 - 750| = 450
To Track 500: |500 - 750| = 250
To Track 900: |900 - 750| = 150
To Track 1500: |1500 - 750| = 750
To Track 300: |300 - 750| = 450
Closest Request: Track 900 (150 units away)

Move to Track 900
Update Position: 900
Total Movement So Far: 150
Remaining Requests: [1200, 500, 1500, 300]

Current Head: 900
Distances to Requests:
To Track 1200: |1200 - 900| = 300
To Track 500: |500 - 900| = 400
To Track 1500: |1500 - 900| = 600
To Track 300: |300 - 900| = 600
Closest Request: Track 1200 (300 units away)

Move to Track 1200
Update Position: 1200
Total Movement So Far: 150 + 300 = 450
Remaining Requests: [500, 1500, 300]

Current Head: 1200

Distances to Requests:
To Track 500: |500 - 1200| = 700
To Track 1500: |1500 - 1200| = 300
To Track 300: |300 - 1200| = 900
Closest Request: Track 1500 (300 units away)

Move to Track 1500
Update Position: 1500
Total Movement So Far: 450 + 300 = 750
Remaining Requests: [500, 300]

Current Head: 1500
Distances to Requests:
To Track 500: |500 - 1500| = 1000
To Track 300: |300 - 1500| = 1200
Closest Request: Track 500 (1000 units away)

Move to Track 500
Update Position: 500
Total Movement So Far: 750 + 1000 = 1750
Remaining Request: [300]

Current Head: 500
Distance to Request: Track 300: |300 - 500| = 200
Move to Track 300
Total Movement So Far: 1750 + 200 = 1950

Summary of Results:
Order of Tracks Visited: [750, 900, 1200, 1500, 500, 300]
Total Head Movement: 1950 units

```
def sstf_disk_scheduling(initial_head, requests):
    # Write code here

# Driver Code
initial_head_position = 750
disk_requests = [1200, 500, 900, 1500, 300]

total_head_movement, tracks_visited = sstf_disk_scheduling(initial_head_position, 
disk_requests)

print("Total Head Movement:", total_head_movement)
print("Order of Tracks Visited:", tracks_visited)
```

8.3 FutureTech Corporation

In the Data Center of FutureTech Corporation, a large disk drive is used to store and retrieve data for various applications. To manage disk access requests efficiently, the disk drive uses the SCAN disk scheduling algorithm. This algorithm is often referred to as the "elevator algorithm" because it works similarly to an elevator in a building: it moves in one direction, servicing requests until it reaches the end, and then reverses direction to service requests on the way back.

The disk has 5000 tracks, numbered from 0 to 4999. The disk head starts at track 2500 and is currently moving in the direction of higher-numbered tracks. Over the course of the day, the following disk access requests are made:

Request 1: Track 2800

Request 2: Track 1500

Request 3: Track 3500

Request 4: Track 4000

Request 5: Track 1000

Problem: Simulate the SCAN algorithm to determine the order in which the disk head will process the requests. Calculate the total head movement required to service all the requests using the SCAN algorithm, given that the disk head is initially moving towards higher-numbered tracks.

Direction of Movement: The disk head starts at track 2500 and is moving towards the higher-numbered tracks. Once it reaches the end of the disk (track 4999), it will reverse direction and service requests on its way back towards track 0.

Requests: Process the requests in the order the head encounters them as it moves in the initial direction and then in the reverse direction.

Head Movement Calculation: Keep track of the total distance moved by the disk head from its initial position to the final position after all requests have been serviced.

Explanation: To solve this problem, follow these steps:
Identify Requests in the Path:
Initial Direction (Higher Tracks):
Start at Track 2500.
Moving towards higher tracks, service the requests that fall within this direction until reaching the end of the disk.
Requests in the Initial Direction:
Track 2800
Track 3500
Track 4000
Reverse Direction (Lower Tracks):
After reaching the end of the disk (Track 4999), reverse direction to service the remaining requests.
Requests in the Reverse Direction:
Track 1500
Track 1000
Simulate the Movement:
Move from Track 2500 to Track 2800: $|2800-2500| = 300$
Move from Track 2800 to Track 3500: $|3500-2800|=700$
Move from Track 3500 to Track 4000: $|4000-3500|=500$
Move from Track 4000 to the End of the Disk (Track 4999): $|4999-4000|=999$
Reverse Direction and Move from Track 4999 to Track 1500: $|1500-4999|=3499$
Move from Track 1500 to Track 1000: $|1000-1500|=500$
Total Head Movement Calculation: $300+700+500+999+3499+500=5498$ units
Order of Tracks Visited:
Start at Track 2500
Move to Track 2800
Move to Track 3500
Move to Track 4000
Move to Track 4999 (end of the disk)
Reverse direction
Move to Track 1500
Move to Track 1000

Summary:
Order of Tracks Visited: [2500, 2800, 3500, 4000, 4999, 1500, 1000]
Total Head Movement: 5498 units

The SCAN algorithm ensures that each request is serviced in an orderly fashion, minimizing the potential delays and ensuring that requests are handled in a systematic manner, similar to how an elevator services floors.

```
def scan_disk_scheduling(initial_head, requests, direction='up', max_track=4999):
    # Write code here

# Driver Code
initial_head_position = 2500
disk_requests = [2800, 1500, 3500, 4000, 1000]

total_head_movement,   tracks_visited   =   scan_disk_scheduling(initial_head_position,
disk_requests, direction='up')

print("Total Head Movement:", total_head_movement)
print("Order of Tracks Visited:", tracks_visited)
```

8.4 The C-SCAN Disk Scheduling Odyssey

In the Advanced Data Center at StellarTech Enterprises, a state-of-the-art disk drive is used to handle high-priority data access requests from various applications. The disk drive uses the Circular SCAN (C-SCAN) disk scheduling algorithm to manage these requests. C-SCAN is designed to provide a more uniform wait time by only servicing requests in one direction and then quickly returning to the beginning of the disk to continue servicing. The disk in the StellarTech data center has 10,000 tracks, numbered from 0 to 9999. The disk head starts at track 4000 and is currently moving in the direction of higher-numbered tracks.

Over the day, the following disk access requests are made:

Request 1: Track 4200

Request 2: Track 1000

Request 3: Track 6000

Request 4: Track 7500

Request 5: Track 2000

Simulate the C-SCAN algorithm to determine the order in which the disk head will process the requests.

Calculate the total head movement required to service all the requests using the C-SCAN algorithm.

Direction of Movement: The disk head starts at track 4000 and is moving towards the higher-numbered tracks. Once it reaches the end of the disk (track 9999), it will quickly move back to track 0 and continue servicing requests from there.

Requests: Process the requests in the direction the head is currently moving (towards higher tracks) and then wrap around to handle requests starting from the beginning of the disk.

Head Movement Calculation: Keep track of the total distance moved by the disk head from its initial position to the final position after all requests have been serviced.

Explanation: To solve this problem, follow these steps:

Identify Requests in the Path:

Initial Direction (Higher Tracks):

Start at Track 4000.

Moving towards higher tracks, service the requests that fall within this direction until reaching the end of the disk.

Requests in the Initial Direction:

Track 4200

Track 6000

Track 7500

Wrap Around:

After reaching the end of the disk (Track 9999), wrap around to the start of the disk (Track 0) to continue servicing the remaining requests.

Requests after Wrapping Around:

Track 1000

Track 2000

Simulate the Movement:

Move from Track 4000 to Track 4200: ∣4200−4000∣=200

Move from Track 4200 to Track 6000: ∣6000−4200∣=1800

Move from Track 6000 to Track 7500: ∣7500−6000∣=1500

Move from Track 7500 to the End of the Disk (Track 9999): ∣9999−7500∣=2499

Wrap Around to Track 0: ∣9999−0∣=9999

Move from Track 0 to Track 1000: ∣1000−0∣=1000

Move from Track 1000 to Track 2000: ∣2000−1000∣=1000

Total Head Movement Calculation: 200+1800+1500+2499+9999+1000+1000=19,998 units

Order of Tracks Visited:

Start at Track 4000

Move to Track 4200

Move to Track 6000

Move to Track 7500

Move to Track 9999 (end of the disk)

Wrap around to Track 0

Move to Track 1000

Move to Track 2000

Order of Tracks Visited: [4000, 4200, 6000, 7500, 9999, 0, 1000, 2000]

Total Head Movement: 19,998 units

The C-SCAN algorithm provides a systematic way of servicing disk requests in one direction and then quickly wrapping around to the beginning to ensure fair treatment of all requests, even those that are far from the current head position.

```
def cscan_disk_scheduling(initial_head, requests, max_track=9999):
    # Write code here

# Driver Code
initial_head_position = 4000
disk_requests = [4200, 1000, 6000, 7500, 2000]
```

```
total_head_movement, tracks_visited = cscan_disk_scheduling(initial_head_position,
disk_requests)

print("Total Head Movement:", total_head_movement)
print("Order of Tracks Visited:", tracks_visited)
```

8.5 The C-SCAN Disk Scheduling Quest at TechFusion Labs

At TechFusion Labs, a cutting-edge research facility, disk drives are crucial for storing and managing vast amounts of scientific data. To efficiently handle disk access requests, the facility uses the Circular SCAN (C-SCAN) disk scheduling algorithm. This algorithm is chosen for its ability to provide fair access to disk resources by continuously moving in one direction and wrapping around to handle all pending requests. The disk drive at TechFusion Labs has 8,000 tracks, numbered from 0 to 7999. The disk head starts at track 3500 and is currently moving towards higher-numbered tracks.
Throughout the day, the following disk access requests are made:
Request A: Track 3800
Request B: Track 600
Request C: Track 7000
Request D: Track 1500
Request E: Track 2500
Simulate the C-SCAN algorithm to determine the order in which the disk head will process the requests. Calculate the total head movement required to service all the requests using the C-SCAN algorithm, given that the disk head starts at track 3500 and moves in the direction of higher-numbered tracks.
Direction of Movement: The disk head starts at track 3500 and is moving towards higher-numbered tracks. Once it reaches the end of the disk (track 7999), it will quickly return to track 0 and continue servicing requests from there.
Requests: Process the requests in the direction the head is currently moving (towards higher tracks) and then wrap around to handle the remaining requests starting from the beginning of the disk.
Head Movement Calculation: Keep track of the total distance moved by the disk head from its initial position to the final position after all requests have been serviced.
Explanation:
Identify Requests in the Path:
Initial Direction (Higher Tracks):
Start at Track 3500.
Moving towards higher tracks, service the requests that fall within this direction until reaching the end of the disk.
Requests in the Initial Direction:
Track 3800
Track 6000
Track 7000
Wrap Around:
After reaching the end of the disk (Track 7999), wrap around to the start of the disk (Track 0) to continue servicing the remaining requests.
Requests after Wrapping Around:
Track 600
Track 1500
Track 2500
Simulate the Movement:
Move from Track 3500 to Track 3800: $|3800-3500|=300$
Move from Track 3800 to Track 6000: $|6000-3800|=2200$
Move from Track 6000 to Track 7000: $|7000-6000|=1000$
Move from Track 7000 to the End of the Disk (Track 7999): $|7999-7000|=999$
Wrap Around to Track 0: $|7999-0|=7999$
Move from Track 0 to Track 600: $|600-0|=600$

Move from Track 600 to Track 1500: |1500−600|=900
Move from Track 1500 to Track 2500: |2500−1500|=1000
Total Head Movement Calculation: 300+2200+1000+999+7999+600+900+1000=14,998 units
Order of Tracks Visited:
Start at Track 3500
Move to Track 3800
Move to Track 6000
Move to Track 7000
Move to Track 7999 (end of the disk)
Wrap around to Track 0
Move to Track 600
Move to Track 1500
Move to Track 2500

Order of Tracks Visited: [3500, 3800, 6000, 7000, 7999, 0, 600, 1500, 2500]
Total Head Movement: 14,998 units

The C-SCAN algorithm ensures that the disk head continuously moves in one direction, wraps around to the beginning of the disk, and processes all pending requests in a fair and systematic manner. This helps in providing uniform wait times and efficient request handling.

```
def cscan_disk_scheduling(initial_head, requests, max_track=7999):
    # Write code here

initial_head_position = 3500
disk_requests = [3800, 600, 7000, 1500, 2500]

total_head_movement, tracks_visited = cscan_disk_scheduling(initial_head_position,
disk_requests)

print("Total Head Movement:", total_head_movement)
print("Order of Tracks Visited:", tracks_visited)
```

9. Concurrency Control

9.1 The Tale of the Library Management System

Imagine a large university library called UniLib, which has an automated system to manage its book inventory and handle user requests. The system must ensure that book information is consistently updated and accessible to all users, even when multiple requests are being processed simultaneously.

- **Book Checkouts:** When a user checks out a book, the system must update the book's availability status.
- **Book Returns:** When a user returns a book, the system must update the book's status to available.
- The library has the following setup:
- **Book Inventory:** A central inventory database that tracks the availability of each book.
- **User Requests:** Users can check out or return books at any time, and multiple requests can be processed concurrently.

The library system faces these challenges:

Concurrency Control: Multiple users may attempt to check out or return the same book simultaneously, leading to potential conflicts and inconsistent book availability information.

Data Integrity: Ensuring that the book’s status is accurately updated and that no two users can perform conflicting operations on the same book.

Challenges:

- **Concurrency Control Mechanisms:** How should the system handle concurrent requests to prevent inconsistencies and ensure that book statuses are correctly updated?

- **Locking Strategies:** What strategies should be used to lock resources (e.g., books) during updates to avoid conflicts and ensure data integrity?
- **Deadlock Prevention:** How can the system avoid deadlocks where users are waiting indefinitely for resources to be released?

Explanation:

To address these challenges, the library system can implement concurrency control techniques and locking strategies:

Concurrency Control Mechanisms:

Use Locking: Employ exclusive locks on books to ensure that only one operation (check out or return) can modify a book's status at a time.

Locking Strategies:

Pessimistic Locking: Lock the book's record when a user starts a checkout or return operation and release the lock only after the operation is completed.

Optimistic Locking: Use version numbers or timestamps to check if the book's status has changed before updating it, rolling back if a conflict is detected.

Deadlock Prevention:

Implement a deadlock prevention strategy such as timeout-based locking where a request is aborted if it cannot acquire a lock within a specified time.

```python
import threading
import time
import random

class BookInventory:
    def __init__(self):
        # Write code here

    def return_book(self, book):
        # Write code here

def user_request(inventory, book, action):
    # Write code here

# Driver Code
inventory = BookInventory()

threads = []
books = ['Book1', 'Book2', 'Book3']
actions = ['checkout', 'return']

for i in range(10):
    book = random.choice(books)
    action = random.choice(actions)
    t = threading.Thread(target=user_request, args=(inventory, book, action))
    threads.append(t)
    t.start()

for t in threads:
    t.join()

print("All operations completed.")
```

9.2 The Tale of the Restaurant Reservation System

Imagine a popular restaurant named Gourmet Haven that uses an automated reservation system to manage table bookings. The restaurant has a limited number of tables, and the reservation system must handle concurrent booking requests efficiently while ensuring that no table is double-booked.
Gourmet Haven operates with:

- **Tables:** The restaurant has 10 tables, each with a unique ID.
- **Reservation System:** A central system that processes reservation requests from multiple customers.

Reservation System faces the following challenges:

- **Concurrency:** Multiple customers might attempt to book the same table at the same time.
- **Data Integrity:** The system must ensure that each table is reserved for only one customer at a time and that all reservation details are correctly updated.

Challenges:

- **Concurrency Control:** How should the system handle simultaneous booking requests to prevent double-booking of tables?
- **Locking Mechanisms:** What strategies should be employed to lock table records during reservation processing to ensure data consistency?
- **Transaction Management:** How can the system manage transactions to ensure that all operations related to a reservation (e.g., updating table status, notifying customers) are completed successfully?

Explanation:
To address these challenges, the reservation system can implement the following strategies:
Concurrency Control Mechanisms:

- **Use Optimistic Locking:** Implement versioning or timestamps to detect if a table's status has changed before finalizing a reservation.
- **Use Pessimistic Locking:** Lock the table’s record while processing the reservation to prevent other requests from modifying it simultaneously.

Locking Strategies:

- **Exclusive Locks:** Lock the table record for the duration of the reservation process to prevent double-booking.

Transaction Management:

- **Atomic Transactions:** Ensure that the reservation process is atomic—either all operations are completed successfully, or none are, rolling back if any part of the transaction fails.

```
import threading
import time
import random

class ReservationSystem:
    def __init__(self, num_tables):
        # Write code here

    def reserve_table(self, table_id):
        # Write code here

    def release_table(self, table_id):
        # Write code here

def customer_request(reservation_system, table_id):
    # Write code here

# Driver Code
reservation_system = ReservationSystem(num_tables=10)

threads = []
```

```
table_ids = [f'Table{i+1}' for i in range(10)]

for _ in range(20):  # Simulate 20 customer requests
    table_id = random.choice(table_ids)
    t = threading.Thread(target=customer_request, args=(reservation_system, table_id))
    threads.append(t)
    t.start()

for t in threads:
    t.join()

print("All reservations completed.")
```

9.3 The Tale of the Online Shopping Cart System

Imagine a popular online shopping platform called ShopEase. The platform has a feature that allows customers to add items to their shopping cart and proceed to checkout. Due to high traffic, the system needs to handle multiple customers accessing and modifying their shopping carts concurrently.
The Problem: ShopEase must ensure:

- **Concurrency:** Multiple customers may attempt to add or remove the same item from their cart simultaneously.
- **Data Integrity:** The system must maintain accurate item counts and prevent inconsistencies in the shopping cart data.

ShopEase operates with:

- **Shopping Carts:** Each customer has a unique shopping cart that can be modified.
- **Inventory System:** The inventory tracks the availability of items and is updated when items are added or removed from carts.

Challenges:

- **Concurrency Control:** How should the system handle simultaneous updates to the same shopping cart or inventory item to prevent data inconsistencies?
- **Locking Mechanisms:** What locking strategies should be employed to ensure that operations on shopping carts and inventory items are synchronized and do not conflict?
- **Transaction Management:** How can the system manage transactions to ensure that all operations related to adding or removing items from the cart are completed successfully or rolled back if any part fails?

Explanation:
To address these challenges, the shopping cart system can implement the following strategies:
Concurrency Control Mechanisms:

- **Pessimistic Locking:** Lock the shopping cart and inventory items while modifications are being made to prevent other operations from conflicting.
- **Optimistic Locking:** Use versioning or timestamps to detect changes in the cart or inventory before committing updates.

Locking Strategies:

- **Exclusive Locks:** Apply exclusive locks on shopping carts and inventory items to ensure that only one modification operation can occur at a time.

Transaction Management:

- **Atomic Transactions:** Ensure that the process of updating the cart and inventory is atomic, meaning all operations must succeed together, or none should apply if there is an error.

```
import threading
import time
import random
```

```python
class InventorySystem:
    def __init__(self):
        # Write code here

    def add_item(self, item, quantity):
        # Write code here

    def remove_item(self, item, quantity):
        # Write code here

class ShoppingCart:
    def __init__(self):
        # Write code here

    def add_to_cart(self, user_id, item, quantity):
        # Write code here

    def remove_from_cart(self, user_id, item, quantity):
        # Write code here

def customer_action(user_id, cart, inventory):
    # Write code here

# Driver Code
inventory = InventorySystem()
cart = ShoppingCart()

threads = []
user_ids = [f'User{i+1}' for i in range(10)]

for user_id in user_ids:
    for _ in range(5):
        t = threading.Thread(target=customer_action, args=(user_id, cart, inventory))
        threads.append(t)
        t.start()

for t in threads:
    t.join()

print("All customer actions completed.")
```

9.4 The Tale of the Collaborative Document Editing System

Imagine a collaborative document editing platform called EditTogether. This platform allows multiple users to edit the same document simultaneously. To maintain consistency and prevent conflicts, the system must manage concurrent access to the document's content effectively.

The Problem: EditTogether needs to handle:

Concurrent Edits: Multiple users might edit the same part of the document at the same time.

Data Integrity: Ensure that all changes are accurately recorded and that no edits are lost or overwritten improperly.

EditTogether operates with:

- **Documents:** Users can open and edit shared documents.
- **Editors:** Each user acts as an editor who can make changes to the document.

Challenges:

- **Concurrency Control:** How should the system handle simultaneous edits to the same section of the document to prevent data corruption and loss?
- **Locking Mechanisms:** What strategies should be employed to lock sections of the document during editing to ensure that changes are applied correctly and consistently?
- **Conflict Resolution:** How can the system detect and resolve conflicts when multiple users attempt to edit the same content concurrently?

Explanation:

To address these challenges, the document editing system can implement the following strategies:

Concurrency Control Mechanisms:

- **Optimistic Locking:** Use version numbers or timestamps to detect if the document or a section has changed since a user started editing. Apply changes only if the document is in the same state as when editing started.
- **Pessimistic Locking:** Lock sections of the document while a user is editing to prevent other users from making changes to the same section.

Locking Strategies:

- **Fine-Grained Locking:** Lock specific sections or paragraphs of the document rather than the entire document to allow multiple users to edit different parts concurrently.
- **Exclusive Locks:** Use exclusive locks on sections being edited to ensure that only one user can make changes at a time.

Conflict Resolution:

- **Change Tracking:** Track changes made by users and detect conflicts when merging changes.
- **Merge Strategies:** Implement merge algorithms to integrate concurrent changes, providing users with options to resolve conflicts manually if necessary.

```
import threading
import time
import random

class Document:
    def __init__(self):
        # Write code here

    def edit_line(self, line_number, new_text):
        # Write code here

    def show_document(self):
        # Write code here

def user_edit(document, user_id):
    # Write code here

# Driver Code
document = Document()

threads = []
user_ids = [f'User{i+1}' for i in range(10)]

for user_id in user_ids:
    t = threading.Thread(target=user_edit, args=(document, user_id))
    threads.append(t)
    t.start()
```

```
for t in threads:
    t.join()

document.show_document()

print("All user edits completed.")
```

9.5 The Tale of the Bank Account Management System

Imagine a large bank called SecureBank that handles thousands of transactions every day. The bank uses an automated system to manage customer accounts. The system must ensure that transactions are processed accurately and that account balances are correctly updated even when multiple transactions occur simultaneously.

SecureBank needs to handle:

- **Concurrent Transactions:** Multiple transactions may be processed on the same account at the same time.
- **Data Integrity:** Ensure that account balances are updated correctly and that no transactions are lost or incorrectly applied.

SecureBank operates with:

- **Accounts:** Each customer has a unique account with a balance.
- **Transactions:** Transactions include deposits and withdrawals that modify account balances.

Challenges:

- **Concurrency Control:** How should the system manage simultaneous transactions on the same account to prevent inconsistencies and ensure correct balance updates?
- **Locking Mechanisms:** What locking strategies should be used to synchronize access to account balances and prevent race conditions?
- **Transaction Management:** How can the system ensure that each transaction is processed atomically and that any failure in the transaction does not leave the account in an inconsistent state?

Explanation:

To address these challenges, the bank system can implement the following strategies:

Concurrency Control Mechanisms:

- **Pessimistic Locking:** Lock the account record while a transaction is being processed to prevent other transactions from modifying the same account concurrently.
- **Optimistic Locking:** Use versioning or timestamps to check if the account balance has changed before applying a transaction, rolling back if a conflict is detected.

Locking Strategies:

- **Exclusive Locks:** Use exclusive locks on account records during transactions to ensure that only one transaction can modify an account's balance at a time.

Transaction Management:

- **Atomic Transactions:** Ensure that each transaction is atomic, meaning that it either fully completes or is fully rolled back if any part fails.

```
import threading
import time
import random

class BankAccount:
    def __init__(self, balance):
        # Write code here

    def deposit(self, amount):
        # Write code here

    def withdraw(self, amount):
        # Write code here

def perform_transaction(account, transaction_type, amount):
    # Write code here

# Driver Code
account = BankAccount(balance=1000)

threads = []
transaction_types = ['deposit', 'withdraw']

for _ in range(20):
    transaction_type = random.choice(transaction_types)
    amount = random.randint(50, 200)
    t = threading.Thread(target=perform_transaction, args=(account, transaction_type,
amount))
    threads.append(t)
    t.start()

for t in threads:
    t.join()
print("All transactions completed.")
```

10. Page Replacement Algorithms

10.1 The Story of a Busy Café and Its Orders

Imagine a cozy café called Café FIFO. This café is known for its efficient handling of orders, thanks to its unique ordering system. The café staff use a FIFO queue to manage their incoming orders, ensuring that the first order received is also the first one to be processed.

Let’s delve into a day at Café FIFO to understand how this system works.

Characters:

- **A**, the café manager.
- **B**, the barista.
- **C**, a customer who places orders.

It’s a bustling morning at Café FIFO. The café has a small counter with a display that can only show a maximum of 4 orders at a time. When more orders come in and the counter is full, the oldest order (the one that’s been there the longest) gets processed and removed to make space for the new ones. This is how the café manages its orders efficiently.

Here's a sequence of orders placed:

Order 1 - Coffee

Order 2 - Tea

Order 3 - Muffin

Order 4 - Croissant

Order 5 - Smoothie

The café processes these orders using the FIFO method, which means:

Order 1 (Coffee) was placed first and is displayed on the counter.

Order 2 (Tea) is next on the counter.

Order 3 (Muffin) is added after Tea.

Order 4 (Croissant) comes in, filling up the counter's capacity of 4.

Now, when Order 5 (Smoothie) arrives, the counter is full. According to FIFO, the oldest order on the counter (Order 1 - Coffee) will be processed and removed to make space for the new order.

To summarize the process:

Order 1 (Coffee) is processed and removed.

Order 5 (Smoothie) is added to the counter.

The counter now has: Order 2 (Tea), Order 3 (Muffin), Order 4 (Croissant), and Order 5 (Smoothie).

At the end of the day, the café's order log shows the following sequence of order arrivals and processing. If you were to check the remaining orders on the counter, which ones would be there and why?

Explanation:

To determine the remaining orders, let's follow the FIFO order replacement process step-by-step:

- Order 1 (Coffee) is processed and removed when Order 5 (Smoothie) arrives.
- The remaining orders after Order 1 is processed are:
    - Order 2 (Tea)
    - Order 3 (Muffin)
    - Order 4 (Croissant)
    - Order 5 (Smoothie)

Thus, at the end of the day, the remaining orders on the counter, in FIFO order, are:

- Order 2 (Tea)
- Order 3 (Muffin)
- Order 4 (Croissant)
- Order 5 (Smoothie)

In this program, we'll maintain a queue to represent the counter in the café, which can hold a maximum of 4 orders at a time. When a new order arrives and the counter is full, the oldest order (the one that arrived first) will be processed and removed to make space for the new order.

```
from collections import deque

class CafeFIFO:
    def __init__(self, capacity):
                # Write code here

    def add_order(self, order):
                # Write code here

    def display_orders(self):
                # Write code here

# Driver Code
```

```
cafe = CafeFIFO(capacity=4)

orders = ["Coffee", "Tea", "Muffin", "Croissant", "Smoothie"]

for order in orders:
    cafe.add_order(order)

cafe.display_orders()
```

10.2 The Tale of the Library and Its Book Shelves

Imagine a small, quaint library called The Library of Timeless Tales. The library has a special section where books are displayed on a shelf with a fixed capacity. The shelf can only hold a maximum of 5 books at a time. When a new book comes in and the shelf is already full, the oldest book on the shelf is removed to make space for the new one.

The librarian, Emma, uses the FIFO policy to manage the books on the shelf. Emma's job is to ensure that the shelf is organized according to the order in which books arrive.

One day, Emma receives a sequence of book requests that need to be displayed on the shelf:

Book A - "The Great Adventure"

Book B - "Mystery of the Lost City"

Book C - "Journey Through Time"

Book D - "Secrets of the Ocean"

Book E - "The Final Frontier"

Book F - "Legends of the Hidden Realm"

Emma needs to display these books on the shelf. Since the shelf can only hold up to 5 books at a time, when a new book arrives and the shelf is full, the oldest book (the one that was on the shelf the longest) must be removed to make room for the new book.

At the end of the day, after all the books have been processed, what books will remain on the shelf?

Explanation:

To solve this problem, we will use a FIFO queue to simulate the shelf management. Here's a step-by-step breakdown:

Book A ("The Great Adventure") is placed on the shelf.

Book B ("Mystery of the Lost City") is added next.

Book C ("Journey Through Time") is added.

Book D ("Secrets of the Ocean") is added.

Book E ("The Final Frontier") is added. At this point, the shelf is full.

When Book F ("Legends of the Hidden Realm") arrives:

Book A (the oldest) will be removed from the shelf.

Book F is then added to the shelf.

After processing all the books, the remaining books on the shelf are:

Book B ("Mystery of the Lost City")

Book C ("Journey Through Time")

Book D ("Secrets of the Ocean")

Book E ("The Final Frontier")

Book F ("Legends of the Hidden Realm")

```python
from collections import deque

class LibraryShelf:
    def __init__(self, capacity):
        # Write code here

    def add_book(self, book):
        # Write code here

    def display_books(self):
        # Write code here

# Driver Code
library_shelf = LibraryShelf(capacity=5)

books = [
    "The Great Adventure",
    "Mystery of the Lost City",
    "Journey Through Time",
    "Secrets of the Ocean",
    "The Final Frontier",
    "Legends of the Hidden Realm"
       ]

for book in books:
    library_shelf.add_book(book)

library_shelf.display_books()
```

10.3 The Story of the Busy Café and Its Special Recipe Book

Imagine a popular café called Café Nostalgia. The café is famous for its unique and varied menu, which changes daily. The chef, Linda, keeps a special recipe book on a counter with a limited space of 4 slots. This book contains recipes that Linda consults frequently, but the counter can only hold up to 4 recipes at a time. When Linda needs to refer to a new recipe and the counter is full, she must remove the recipe that hasn't been used for the longest time to make space for the new one. One day, Linda receives a sequence of recipe requests. She uses the LRU (Least Recently Used) policy to manage her recipe book on the counter.

Linda receives the following sequence of recipe requests for the day:

Recipe 1 - "Pumpkin Spice Latte"

Recipe 2 - "Apple Pie"

Recipe 3 - "Blueberry Muffin"

Recipe 4 - "Cinnamon Roll"

Recipe 2 - "Apple Pie"

Recipe 5 - "Maple Pancakes"

Recipe 1 - "Pumpkin Spice Latte"

Linda's job is to ensure that the recipe book reflects the most recently used recipes according to the LRU policy.

At the end of the day, after processing all the recipe requests, which recipes will remain on the counter, and in what order?

Explanation:

To solve this problem, we need to simulate the LRU page replacement algorithm. The LRU policy keeps track of the usage order of recipes, removing the least recently used recipe when the counter is full.

Here’s a step-by-step breakdown of how the recipes will be managed:

Recipe 1 ("Pumpkin Spice Latte") is added to the counter.

Recipe 2 ("Apple Pie") is added.

Recipe 3 ("Blueberry Muffin") is added.

Recipe 4 ("Cinnamon Roll") is added. At this point, the counter is full.

When Recipe 2 ("Apple Pie") is requested again:

Since Recipe 2 is already on the counter and it’s the most recently used, it remains on the counter.

The state of the counter remains: Recipe 1, Recipe 2, Recipe 3, Recipe 4.

When Recipe 5 ("Maple Pancakes") is requested:

The counter is full, so the least recently used recipe, Recipe 1 ("Pumpkin Spice Latte"), is removed.

Recipe 5 is added to the counter.

The state of the counter becomes: Recipe 2, Recipe 3, Recipe 4, Recipe 5.

When Recipe 1 ("Pumpkin Spice Latte") is requested again:

Recipe 1 is no longer on the counter, so it is re-added, and the least recently used recipe, Recipe 3 ("Blueberry Muffin"), is removed.

The final state of the counter becomes: Recipe 2, Recipe 4, Recipe 5, Recipe 1.

```python
from collections import OrderedDict

class CafeRecipeBook:
    def __init__(self, capacity):
        # Write code here

    def add_recipe(self, recipe):
        # Write code here

    def display_recipes(self):
        # Write code here

# Driver Code
cafe_recipe_book = CafeRecipeBook(capacity=4)

recipes = [
    "Pumpkin Spice Latte",
    "Apple Pie",
    "Blueberry Muffin",
    "Cinnamon Roll",
    "Apple Pie",
    "Maple Pancakes",
    "Pumpkin Spice Latte"
          ]

for recipe in recipes:
    cafe_recipe_book.add_recipe(recipe)

cafe_recipe_book.display_recipes()
```

10.4 The Tale of the Art Gallery and Its Exhibition

Imagine a chic art gallery named Gallery of Modern Arts. The gallery features a special digital display board that can showcase up to 6 artworks at a time. The display board is updated frequently to reflect the most popular and recent artworks. Sophia, the gallery curator, manages the display using the LRU (Least Recently Used) policy. When a new artwork needs to be showcased and the board is already full, Sophia must remove the artwork that has been displayed the longest and add the new one.

Sophia receives the following sequence of artwork requests throughout the day:

Artwork 1 - "Sunset Over the Lake"

Artwork 2 - "Abstract Dreams"

Artwork 3 - "Cityscape at Dusk"

Artwork 4 - "Portrait of the Artist"

Artwork 5 - "Golden Fields"

Artwork 6 - "Mystic Mountains"

Artwork 2 - "Abstract Dreams"

Artwork 7 - "Twilight Reflections"

Artwork 1 - "Sunset Over the Lake"

Artwork 8 - "Harmony in Blue"

Sophia needs to manage the display board such that it shows the most recently used artworks according to the LRU policy. At the end of the day, after processing all the artwork requests, which artworks will remain on the display board, and in what order?

Explanation:

To solve this problem using the LRU page replacement algorithm, we'll simulate how the display board is updated. The LRU policy ensures that the least recently used artwork is removed when the board is full.

Here's a step-by-step breakdown:

Artwork 1 ("Sunset Over the Lake") is added to the display board.

Artwork 2 ("Abstract Dreams") is added.

Artwork 3 ("Cityscape at Dusk") is added.

Artwork 4 ("Portrait of the Artist") is added.

Artwork 5 ("Golden Fields") is added.

Artwork 6 ("Mystic Mountains") is added. At this point, the display board is full.

When Artwork 2 ("Abstract Dreams") is requested again:

Artwork 2 is already on the board and is moved to the most recently used position.

When Artwork 7 ("Twilight Reflections") is requested:

The board is full, so the least recently used artwork, Artwork 3 ("Cityscape at Dusk"), is removed.

Artwork 7 is added to the board.

When Artwork 1 ("Sunset Over the Lake") is requested:

Artwork 1 is already on the board and is moved to the most recently used position.

When Artwork 8 ("Harmony in Blue") is requested:

The board is full, so the least recently used artwork, Artwork 4 ("Portrait of the Artist"), is removed.

Artwork 8 is added to the board.

The final state of the display board, after processing all artwork requests, will be:

Artwork 2 ("Abstract Dreams")

Artwork 7 ("Twilight Reflections")

Artwork 1 ("Sunset Over the Lake")

Artwork 5 ("Golden Fields")

Artwork 6 ("Mystic Mountains")

Artwork 8 ("Harmony in Blue")

```python
from collections import OrderedDict

class ArtGalleryDisplay:
    def __init__(self, capacity):
        # Write code here

    def add_artwork(self, artwork):
        # Write code here

    def display_artworks(self):
        # Write code here

# Driver Code
art_gallery_display = ArtGalleryDisplay(capacity=6)

artworks = [
    "Sunset Over the Lake",
    "Abstract Dreams",
    "Cityscape at Dusk",
    "Portrait of the Artist",
    "Golden Fields",
    "Mystic Mountains",
    "Abstract Dreams",
    "Twilight Reflections",
    "Sunset Over the Lake",
    "Harmony in Blue"
           ]

for artwork in artworks:
    art_gallery_display.add_artwork(artwork)

art_gallery_display.display_artworks()
```

10.5 The Tale of the Library and Its Popular Books

Imagine a charming library called The Book Haven. The library has a special digital display that showcases the top 5 most popular books at any given time. The display is updated based on how frequently each book is checked out. Liam, the library manager, uses the LFU (Least Frequently Used) policy to manage the display. If a new book needs to be featured on the display and the display is already full, Liam must remove the book that is checked out the least number of times to make room for the new one.

Throughout the day, the library receives the following sequence of book checkouts:

Book A - "The Great Gatsby" (Checked out 3 times)

Book B - "1984" (Checked out 5 times)

Book C - "To Kill a Mockingbird" (Checked out 2 times)

Book D - "Pride and Prejudice" (Checked out 4 times)

Book E - "The Catcher in the Rye" (Checked out 1 time)

Book F - "Moby Dick" (Checked out 6 times)

Book G - "The Odyssey" (Checked out 3 times)

Book H - "War and Peace" (Checked out 2 times)

Liam needs to manage the display so that it shows the most frequently checked out books according to the LFU policy. At the end of the day, after processing all the book checkouts, which books will remain on the display, and in what order?

Explanation:

To solve this problem using the LFU page replacement algorithm, we need to keep track of how frequently each book is checked out. When the display is full, the least frequently checked out book (or the one with the lowest frequency) should be removed to make space for the new book.

Here’s a step-by-step breakdown of how the books will be managed:

Book A ("The Great Gatsby") is added to the display.

Book B ("1984") is added.

Book C ("To Kill a Mockingbird") is added.

Book D ("Pride and Prejudice") is added.

Book E ("The Catcher in the Rye") is added. At this point, the display is full.

When Book F ("Moby Dick") is checked out:

Book E ("The Catcher in the Rye") is the least frequently checked out book (checked out only 1 time). It is removed to make room for Book F.

Book F is added to the display.

When Book G ("The Odyssey") is checked out:

Book C ("To Kill a Mockingbird") and Book H ("War and Peace") both have a frequency of 2. However, Book C was removed earlier, so it has a higher chance of being added back.

Book G is added to the display, and Book H is added as well, replacing the least frequently checked out book among those with the same frequency.

The final state of the display, considering the frequencies of checkouts, will be:

Book B ("1984") - 5 times

Book F ("Moby Dick") - 6 times

Book D ("Pride and Prejudice") - 4 times

Book G ("The Odyssey") - 3 times

Book H ("War and Peace") - 2 times

```
from collections import defaultdict, Counter
import heapq
```

```
class LibraryDisplay:
    def __init__(self, capacity):
        # Write code here

    def add_book(self, book, times_checked_out):
        # Write code here

    def display_books(self):
        # Write code here

# Driver Code
library_display = LibraryDisplay(capacity=5)

book_checkouts = [
    ("The Great Gatsby", 3),
    ("1984", 5),
    ("To Kill a Mockingbird", 2),
    ("Pride and Prejudice", 4),
    ("The Catcher in the Rye", 1),
    ("Moby Dick", 6),
    ("The Odyssey", 3),
    ("War and Peace", 2)
]

for book, times_checked_out in book_checkouts:
    library_display.add_book(book, times_checked_out)

library_display.display_books()
```

11. Process Synchronization

11.1 The Tale of the Bakery and Its Busy Kitchen

Imagine a bustling bakery named Sweet Delights. The bakery is renowned for its delicious pastries and cakes, which are made in a busy kitchen. The kitchen has a single oven that can only bake one item at a time. The bakery operates with multiple bakers working simultaneously, and they need to use the oven to bake their items. Lily, the head baker, needs to ensure that the oven is used efficiently and fairly. She uses a process synchronization mechanism to manage access to the oven so that no two bakers use it at the

In the bakery, there are three bakers: A, B, and C. They need to bake the following items:

A wants to bake a Chocolate Cake.

B wants to bake a Fruit Tart.

C wants to bake a Cheese Croissant.

Each baker will request access to the oven, bake their item, and then release the oven for the next baker.

Challenges:

Mutual Exclusion: Only one baker should be able to use the oven at any given time.

Progress: If no baker is using the oven, and there are bakers who want to use it, then the next baker should get access to it.

Fairness: Every baker should get a chance to use the oven in the order they requested.

Explanation:

To solve this problem, Lily can use a semaphore-based synchronization mechanism to ensure that these conditions are met. Here's how she can implement this:

Initialize a semaphore to control access to the oven. The semaphore will be initialized to 1 to ensure mutual exclusion. Implement the process synchronization so that when a baker wants to use the oven, they wait (i.e., block) if the oven is already in use, and proceed only when they are granted access. Use a queue or list to keep track of the order in which bakers request access to the oven to ensure fairness.

```python
import threading
import time
from queue import Queue

oven_semaphore = threading.Semaphore(1)

request_queue = Queue()

def baker(name, bake_time):
    # Write code here

# Driver Code
baker_threads = [
    threading.Thread(target=baker, args=("A", 2)),
    threading.Thread(target=baker, args=("B", 3)),
    threading.Thread(target=baker, args=("C", 1))
]

for thread in baker_threads:
    thread.start()

for thread in baker_threads:
    thread.join()

print("All bakers have finished using the oven.")
```

11.2 The Tale of the Busy Coffee Shop and Its Coffee Machines

Imagine a popular coffee shop called Brewed Bliss. The coffee shop has two coffee machines, Machine 1 and Machine 2, but only one machine can be used at a time to prepare a customer's order. The shop is busy with several baristas working simultaneously, and they need to use the machines to prepare coffee drinks for customers. John, the coffee shop manager, needs to ensure that the coffee machines are used efficiently and fairly. He uses a process synchronization mechanism to manage access to the coffee machines so that no two baristas use the same machine at the same time, and each barista gets a chance to use the machines in an orderly manner. In the coffee shop, there are three baristas: Emma, Liam, and Olivia. Each barista has the following orders to prepare:

- Emma wants to use Machine 1 to prepare a Latte.
- Liam wants to use Machine 2 to prepare a Cappuccino.
- Olivia wants to use Machine 1 to prepare an Espresso.

Each barista will request access to the coffee machines, use them to prepare their drink, and then release the machines for the next barista.

Challenges:

Mutual Exclusion: Only one barista should be able to use a particular coffee machine at any given time.

Progress: If a coffee machine is free and there are baristas waiting to use it, then the next barista should be able to use the machine.

Fairness: Each barista should get a fair chance to use the coffee machines in the order they requested.

Explanation:

To solve this problem, John can use semaphores and a queue-based synchronization mechanism to ensure that these conditions are met. Here's a step-by-step approach:

Initialize semaphores to control access to each coffee machine. Each semaphore will be initialized to 1 to ensure mutual exclusion for each machine.

Implement a queue to manage the order in which baristas request access to the coffee machines to ensure fairness. Ensure that when a barista finishes using a machine, the next barista in the queue gets a chance to use it, maintaining fairness.

```python
import threading
import time
from collections import deque

machine_1_semaphore = threading.Semaphore(1)
machine_2_semaphore = threading.Semaphore(1)

request_queue = deque()

queue_lock = threading.Lock()

def barista(name, machine, preparation_time):
    # Write code here

# Driver Code
barista_threads = [
    threading.Thread(target=barista, args=("Emma", "Machine 1", 2)),
    threading.Thread(target=barista, args=("Liam", "Machine 2", 3)),
    threading.Thread(target=barista, args=("Olivia", "Machine 1", 1))
                  ]

for thread in barista_threads:
    thread.start()

for thread in barista_threads:
    thread.join()

print("All baristas have finished using the coffee machines.")
```

11.3 The Tale of the Conference Room and Its Reservations

Imagine a company called Tech Innovations, which has a highly sought-after conference room for meetings and presentations. The conference room can be reserved by employees for various meetings, but it can only be used by one employee at a time. Sarah, the office manager, needs to manage the reservations for the conference room to ensure that no two employees are double-booked and that the reservations are handled in a fair manner. She uses a synchronization mechanism to handle access to the conference room so that no two employees can reserve the room at the same time.

In the company, there are three employees: A, B, and C. Each employee wants to reserve the conference room for a meeting at different times:

A wants to reserve the conference room from 10:00 AM to 11:00 AM.

B wants to reserve the conference room from 11:00 AM to 12:00 PM.

C wants to reserve the conference room from 12:00 PM to 1:00 PM.

Each employee will request access to the conference room, use it during their reserved time, and then release it for the next employee.

Challenges:

Mutual Exclusion: Only one employee should be able to use the conference room at any given time.

Progress: If the conference room is free and there are employees waiting to use it, then the next employee should be granted access to it.

Fairness: Each employee should get the conference room for the time they reserved, in the order of their reservation.

Explanation:

To solve this problem, Sarah can use semaphores and a queue-based synchronization mechanism to ensure that these conditions are met. Here's how it can be implemented:

- Initialize a semaphore to control access to the conference room. The semaphore will be initialized to 1 to ensure mutual exclusion.
- Implement a queue to manage the order in which employees request the conference room to ensure fairness.
- Ensure that when an employee finishes using the conference room, the next employee in the queue gets access to it.

```
import threading
import time
from collections import deque

conference_room_semaphore = threading.Semaphore(1)

request_queue = deque()

queue_lock = threading.Lock()

def employee(name, start_time, end_time):
    # Write code here


# Driver Code
employee_threads = [
    threading.Thread(target=employee, args=("A", "10:00", "11:00")),
    threading.Thread(target=employee, args=("B", "11:00", "12:00")),
    threading.Thread(target=employee, args=("C", "12:00", "13:00"))
]

for thread in employee_threads:
    thread.start()

for thread in employee_threads:
    thread.join()

print("All employees have finished using the conference room.")
```

11.4 The Tale of the Restaurant Kitchen and Its Limited Resources

Imagine a busy restaurant called Gourmet Haven. The kitchen in this restaurant is equipped with a limited number of critical resources for cooking: one stove and one refrigerator. Only one cook can use the stove at a time to prepare dishes, and only one cook can access the refrigerator at a time to retrieve ingredients. Tom, the head chef, needs to ensure that the cooks can use these resources efficiently and fairly. He uses a process synchronization mechanism to manage access to the stove and refrigerator so that no two cooks use the same resource at the same time, and each cook gets a fair chance to use the resources.

In the kitchen, there are three cooks: J, M, and R. They each need to use the stove and refrigerator for their cooking tasks:

J needs to use the stove to prepare a pasta and the refrigerator to get some cheese.

M needs to use the stove to prepare a stir-fry and the refrigerator to get some vegetables.

R needs to use the stove to prepare a soup and the refrigerator to get some herbs.

Each cook will request access to the stove and refrigerator, use them for their tasks, and then release the resources for the next cook.

Challenges:

Mutual Exclusion: Only one cook should be able to use the stove or the refrigerator at any given time.

Progress: If a resource is free and there are cooks waiting to use it, then the next cook should be granted access.

Fairness: Each cook should get access to both the stove and refrigerator in the order they requested, without causing deadlock.

Explanation:

To solve this problem, Tom can use semaphores and a queue-based synchronization mechanism to ensure that these conditions are met. Here's a step-by-step approach:

- Initialize semaphores to control access to the stove and refrigerator. Each semaphore will be initialized to 1 to ensure mutual exclusion.
- Implement a queue to manage the order in which cooks request access to the resources to ensure fairness.
- Ensure that when a cook finishes using a resource, the next cook in the queue gets access to it, maintaining fairness and preventing deadlock.

```
import threading
import time
from collections import deque

stove_semaphore = threading.Semaphore(1)
refrigerator_semaphore = threading.Semaphore(1)

request_queue = deque()

queue_lock = threading.Lock()

def cook(name, stove_time, refrigerator_time):
    # Write code here


# Driver Code
cook_threads = [
    threading.Thread(target=cook, args=("J", 2, 1)),
    threading.Thread(target=cook, args=("M", 3, 2)),
    threading.Thread(target=cook, args=("R", 1, 1))
              ]

for thread in cook_threads:
    thread.start()

for thread in cook_threads:
    thread.join()
```

```
print("All cooks have finished using the stove and refrigerator.")
```

11.5 The Tale of the Garden and Its Watering Schedule

Imagine a community garden named Green Oasis. The garden has a single water pump that can be used to water the plants. Due to the importance of keeping the plants well-watered, the garden has established strict guidelines to ensure that only one gardener uses the water pump at a time. Linda, the head gardener, needs to manage the use of the water pump so that no two gardeners use it simultaneously and that each gardener gets their turn to water their plants. She uses a process synchronization mechanism to handle access to the water pump fairly and efficiently.

In the garden, there are three gardeners: S, J, and O. They each need to use the water pump for different periods to water their assigned garden plots:

S needs to use the water pump to water her plot for 30 minutes.

J needs to use the water pump to water his plot for 45 minutes.

O needs to use the water pump to water his plot for 20 minutes.

Each gardener will request access to the water pump, use it for their designated time, and then release it for the next gardener.

Challenges:

Mutual Exclusion: Only one gardener should be able to use the water pump at any given time.

Progress: If the water pump is free and there are gardeners waiting to use it, then the next gardener should be granted access.

Fairness: Each gardener should get access to the water pump in the order they requested, without causing deadlock.

Explanation:

To solve this problem, Linda can use semaphores and a queue-based synchronization mechanism to ensure that these conditions are met. Here's a step-by-step approach:

- Initialize a semaphore to control access to the water pump. The semaphore will be initialized to 1 to ensure mutual exclusion.
- Implement a queue to manage the order in which gardeners request access to the water pump to ensure fairness.
- Ensure that when a gardener finishes using the water pump, the next gardener in the queue gets access to it.

```
import threading
import time
from collections import deque

water_pump_semaphore = threading.Semaphore(1)

request_queue = deque()

queue_lock = threading.Lock()
```

```
def gardener(name, water_time):
    # Write code here

# Driver Code
gardener_threads = [
    threading.Thread(target=gardener, args=("Sophia", 30)),
    threading.Thread(target=gardener, args=("James", 45)),
    threading.Thread(target=gardener, args=("Oliver", 20))
]

for thread in gardener_threads:
    thread.start()
for thread in gardener_threads:
    thread.join()

print("All gardeners have finished using the water pump.")
```

12. Deadlock and Prevention

12.1 The Tale of the Printing Press and Its Inks

Imagine a large printing company called Print Master, which produces high-quality magazines and brochures. The company operates two printing presses: Press 1 and Press 2. Each press needs to use multiple types of inks to print various designs. However, each press can only use one ink type at a time. E and L, two senior printers at Print Master, often need to use the presses and inks for different projects. They face a problem with their printing operations due to resource contention, leading to potential deadlocks.

- E and L both need to use the presses and inks simultaneously for their respective projects:
- E needs Press 1 and Ink A to print a brochure and Press 2 and Ink B to print a flyer.
- L needs Press 2 and Ink B to print a poster and Press 1 and Ink A to print a catalog.

Due to the constraints, there is a chance of a deadlock scenario where neither Emma nor Lucas can proceed. For instance, if Emma holds Press 1 and Ink A while waiting for Press 2 and Ink B, and Lucas holds Press 2 and Ink B while waiting for Press 1 and Ink A, both will be stuck waiting indefinitely.

Challenges:

Deadlock Prevention: Ensure that the system is designed to avoid situations where both Emma and Lucas end up waiting for each other's resources indefinitely.

Resource Allocation: Manage the allocation of presses and inks such that deadlock situations are identified and handled efficiently.

Explanation:

To handle this problem, Print Master can use strategies to prevent deadlocks, such as resource allocation strategies and introducing rules to avoid circular wait conditions.

Strategy to Avoid Deadlock:

- **Introduce a Resource Allocation Protocol:** Create a protocol that ensures resources are allocated in a way that prevents deadlock. For instance, ensuring that both Emma and Lucas must request all resources they need at once.
- **Implement Resource Request and Release Management**: Use a system to track resource requests and releases to ensure that if a resource request cannot be fulfilled, the system can rollback or preempt resources.

```
import threading
import time
```

```
press_1_lock = threading.Lock()
press_2_lock = threading.Lock()
ink_a_lock = threading.Lock()
ink_b_lock = threading.Lock()

def use_resources(printer_name, need_press_1, need_press_2, need_ink_a, need_ink_b):
    # Write code here

# Driver Code
e_thread = threading.Thread(target=use_resources, args=("E", True, True, True, True))
l_thread = threading.Thread(target=use_resources, args=("L", True, True, True, True))

e_thread.start()
l_thread.start()

e_thread.join()
l_thread.join()

print("All jobs have been completed.")
```

12.2 The Tale of the Library and Its Book Reservations

Imagine a popular public library called Book Haven. The library has a special Rare Books Room with two highly sought-after books: Book A and Book B. Due to the rarity of these books, only one person can reserve each book at a time. Additionally, each person must use a special reading desk in the Rare Books Room. S and E, two avid readers, both want to read these rare books, but their requests might lead to a deadlock situation if not managed properly.

S and E both need access to the following resources:

- **S** needs Book A and the reading desk for 2 hours.
- **E** needs Book B and the reading desk for 1.5 hours.

Due to the constraints, there is a risk of a deadlock scenario. For example:

S might hold Book A and the reading desk while waiting for Book B.

E might hold Book B and the reading desk while waiting for Book A.

If both are waiting for each other's resources, neither can proceed, leading to a deadlock.

Challenges:

- **Deadlock Prevention:** Ensure that the reservation system avoids scenarios where both Sophia and Ethan end up waiting for each other's resources indefinitely.
- **Resource Allocation:** Manage the allocation of books and desks so that the library can handle requests efficiently without falling into deadlock.

Explanation:

To handle this problem, Book Haven can implement a strategy to prevent deadlocks by ensuring that resources are requested and allocated in a way that avoids circular wait conditions.

Strategy to Avoid Deadlock:

- **Request All Resources at Once:** Ensure that a reservation request is only granted if all required resources (books and desk) are available.
- **Resource Allocation Protocol:** Use a system where a person can only request the desk if both books are available, or request both books if the desk is available.

```
import threading
import time

book_a_lock = threading.Lock()
book_b_lock = threading.Lock()
desk_lock = threading.Lock()

def reserve_resources(reader_name, need_book_a, need_book_b, need_desk, reading_time):
    # Write code here

# Driver Code
s_thread = threading.Thread(target=reserve_resources, args=("S", True, False, True,
2))
e_thread = threading.Thread(target=reserve_resources, args=("E", False, True, True,
1.5))

sophia_thread.start()
ethan_thread.start()

sophia_thread.join()
ethan_thread.join()

print("All reservations have been completed.")
```

12.3 The Tale of the Automated Warehouse and Its Robots

In a high-tech automated warehouse called SmartStore, there are multiple robots responsible for moving items from shelves to shipping areas. The warehouse has two critical resources: Robot A and Robot B. Each robot can operate independently, but there are times when both robots need to access specific storage bins to complete their tasks. A and B, two warehouse managers, are in charge of scheduling tasks for the robots. They need to ensure that the robots are used efficiently to avoid conflicts that could lead to deadlocks.

A and B have tasks that require the following resources:

- A needs Robot A to pick up items from Bin 1 and Robot B to pick up items from Bin 2.
- B needs Robot B to pick up items from Bin 2 and Robot A to pick up items from Bin 1.

If A starts her task with Robot A and Robot B, and B starts his task with Robot B and Robot A, a deadlock situation may occur where each manager waits indefinitely for the other robot to be freed.

Challenges:

- **Deadlock Prevention:** Ensure that the system is designed to avoid situations where both Alice and Bob end up waiting for each other's robots indefinitely.
- **Resource Allocation:** Manage the allocation of robots to ensure that their tasks are completed efficiently without deadlock.

Explanation:

To handle this problem, SmartStore can use a strategy to avoid deadlocks by ensuring that requests for resources do not lead to circular waiting conditions.

Strategy to Avoid Deadlock:

- **Resource Request Ordering:** Establish a protocol that all tasks must follow in which resources are requested in a consistent order to prevent circular wait conditions.
- **Avoid Hold and Wait:** Ensure that if a task requests additional resources, it must release its currently held resources if the required resources are not available.

```
import threading
import time

robot_a_lock = threading.Lock()
robot_b_lock = threading.Lock()
bin_1_lock = threading.Lock()
bin_2_lock = threading.Lock()

def task(name, need_robot_a, need_robot_b, need_bin_1, need_bin_2, work_time):
    # Write code here

# Driver Code
a_thread = threading.Thread(target=task, args=("A", True, True, True, True, 3))
b_thread = threading.Thread(target=task, args=("B", True, True, True, True, 2))

alice_thread.start()
bob_thread.start()

a_thread.join()
b_thread.join()

print("All tasks have been completed.")
```

12.4 The Tale of the Coffee Shop and Its Baristas

Imagine a popular coffee shop named Java Junction. The shop has two crucial resources that its baristas need to prepare drinks: Espresso Machine and Blender. These resources are shared, and only one barista can use each resource at a time. A and B, the baristas, have overlapping shifts and sometimes need to use both the Espresso Machine and the Blender simultaneously to prepare complex coffee orders.
A and B each have orders that require the following resources:

- A needs the Espresso Machine to make a Latte and the Blender to make a Smoothie.
- B needs the Blender to make a Frappuccino and the Espresso Machine to make an Espresso.

Due to their overlapping schedules, if Anna starts using the Espresso Machine and Ben starts using the Blender, a deadlock situation might occur:

- A might hold the Espresso Machine and be waiting for the Blender.
- B might hold the Blender and be waiting for the Espresso Machine.

Both baristas end up waiting indefinitely for the resources held by the other, causing a deadlock.

Challenges:

- **Deadlock Prevention:** Ensure that the resource allocation system avoids scenarios where both Anna and Ben are waiting for each other's resources indefinitely.
- **Resource Allocation:** Manage the allocation of the Espresso Machine and Blender to avoid deadlock and ensure efficient operation.

Explanation:
To handle this problem, Java Junction can implement a strategy to avoid deadlocks by ensuring that resources are requested and used in a way that avoids circular dependencies.

Strategy to Avoid Deadlock:

- **Resource Ordering:** Establish a rule that resources must be requested in a specific order, ensuring that circular wait conditions are avoided.

- **Resource Allocation Protocol:** Create a protocol where a barista must request all needed resources at once and will only proceed if all are available.

```
import threading
import time

espresso_machine_lock = threading.Lock()
blender_lock = threading.Lock()

def make_drink(barista_name, need_espresso_machine, need_blender, work_time):
    # Write code here

# Driver Code
a_thread = threading.Thread(target=make_drink, args=("A", True, True, 3))
b_thread = threading.Thread(target=make_drink, args=("B", True, True, 2))

anna_thread.start()
ben_thread.start()

anna_thread.join()
ben_thread.join()

print("All drinks have been prepared.")
```

12.5 The Tale of the Theater Production and Its Props

Imagine a local theater company named Stagecraft Productions. They are preparing for a major play and have two critical props that need to be managed: Prop X and Prop Y. Each prop is essential for different scenes and can only be used by one actor at a time. E and L, two actors, are preparing for their roles. They each need to use both props to rehearse their scenes. Emma and Liam's rehearsal schedules overlap, which might lead to a potential deadlock situation.

E and L have the following prop requirements:

- **E** needs Prop X to rehearse her Scene 1 and Prop Y to rehearse her Scene 2.
- **L** needs Prop Y to rehearse his Scene 3 and Prop X to rehearse his Scene 4.

If E starts rehearsing with Prop X and L starts rehearsing with Prop Y, they may end up in a deadlock:

- **E** might hold Prop X while waiting for Prop Y.
- **L** might hold Prop Y while waiting for Prop X.

This leads to a situation where both actors are waiting indefinitely for the props held by the other.

Challenges:

- **Deadlock Prevention:** Ensure that the rehearsal system avoids scenarios where both Emma and Liam are waiting for each other's props indefinitely.
- **Resource Allocation:** Manage the allocation of props to ensure efficient rehearsal schedules without deadlock.

Explanation:

To handle this problem, Stagecraft Productions can implement strategies to avoid deadlocks by ensuring that the props are requested and used in a way that avoids circular dependencies.

Strategy to Avoid Deadlock:

- **Resource Request Protocol:** Ensure that props are requested in a specific order, and that each actor must request all required props at once.
- **Avoid Hold and Wait:** Implement a policy where if an actor cannot get all the props they need, they must release any currently held props and retry.

```
import threading
import time

prop_x_lock = threading.Lock()
prop_y_lock = threading.Lock()

def rehearse(actor_name, need_prop_x, need_prop_y, rehearsal_time):
    # Write code here

# Driver Code
e_thread = threading.Thread(target=rehearse, args=("E", True, True, 3))
l_thread = threading.Thread(target=rehearse, args=("L", True, True, 2))

e_thread.start()
l_thread.start()

e_thread.join()
l_thread.join()

print("All rehearsals have been completed.")
```

13. Queuing Theory

13.1 Customer Service - M/M/1 Queue Analysis

You are a manager at a customer service center for a popular online retailer. Customers call in to inquire about their orders, request assistance with returns, or seek product information. Your task is to analyze the customer service operations using the M/M/1 queuing model to understand and optimize service efficiency.

At your customer service center:

- Customers arrive randomly and independently at an average rate of 10 customers per hour.
- Each customer's service request, on average, takes 6 minutes to resolve.
- There is one customer service representative available to handle customer inquiries.

Objective:

Model Selection: Apply the M/M/1 queuing model to analyze the customer service operations.

Performance Metrics: Calculate and analyze key metrics such as:

Average number of customers waiting in queue.

Average time a customer spends in the system (waiting + service time).

Utilization of the customer service representative.

Optimization Recommendations: Based on the analysis, propose improvements to enhance customer service efficiency.

Tasks to Implement:

Model Formulation: Use the M/M/1 queuing model formulas to calculate steady-state metrics.

Data Collection: Gather real-time or historical data on customer arrival rates and service times.

Analysis and Recommendations: Interpret queuing model outputs to identify bottlenecks and suggest operational improvements.

Implement functions or formulas to simulate customer arrivals and service times based on provided parameters. Use the M/M/1 queuing model to predict metrics such as average number of customers in queue and average customer waiting time. Evaluate scenarios (e.g., adjusting staffing levels) to optimize service efficiency based on queuing theory insights.

```
def mm1_queue(arrival_rate, service_time):
    # Write code here

# Driver Code
arrival_rate = 10 / 60
service_time = 6

avg_customers, avg_time_in_system, utilization = mm1_queue(arrival_rate, service_time)

print(f"M/M/1 Queue - Average Customers: {avg_customers:.2f}, Average Time in System: 
{avg_time_in_system:.2f} minutes, Utilization: {utilization:.2f}")
```

13.2 Telecommunication Company - M/M/c Queue Analysis

You are a manager at a busy telecommunications company that provides technical support to customers over the phone. Customers call in for assistance with network issues, billing inquiries, and service upgrades. Your task is to analyze the efficiency of your customer support operations using the M/M/c queuing model.

At your customer support center:

- Customers arrive randomly and independently at an average rate of 15 customers per hour.
- Each customer's service request, on average, takes 8 minutes to resolve.
- Your center has 3 customer service representatives available to handle customer inquiries simultaneously.

Objective:

Model Selection: Apply the M/M/c queuing model to analyze the customer support operations.

Performance Metrics: Calculate and analyze key metrics such as:

Average number of customers waiting in queue.

Average time a customer spends in the system (waiting + service time).

Utilization of the customer service representatives.

Optimization Recommendations: Based on the analysis, propose improvements to enhance customer service efficiency.

Tasks to Implement:

Model Formulation: Use the M/M/c queuing model formulas to calculate steady-state metrics.

Data Collection: Gather real-time or historical data on customer arrival rates and service times.

Analysis and Recommendations: Interpret queuing model outputs to identify bottlenecks and suggest operational improvements.

Implement functions or formulas to simulate customer arrivals and service times based on provided parameters. Use the M/M/c queuing model to predict metrics such as average number of customers in queue and average customer waiting time. Evaluate scenarios (e.g., adjusting staffing levels) to optimize service efficiency based on queuing theory insights.

```
def mmc_queue(arrival_rate, service_time, num_servers):
    # Write code here

# Driver Code
import math

arrival_rate = 15 / 60  # customers per minute
service_time = 8  # minutes
num_servers = 3

avg_customers, avg_time_in_system, utilization = mmc_queue(arrival_rate, service_time,
num_servers)

print(f"M/M/{num_servers} Queue - Average Customers: {avg_customers:.2f}, Average Time
in System: {avg_time_in_system:.2f} minutes, Utilization: {utilization:.2f}")
```

13.3 The Tale of the Busy Call Center and Its Call Routing System

Imagine a bustling call center named QuickConnect, which handles customer support for a large telecommunications company. The call center uses a sophisticated call routing system to manage incoming customer calls. The system's goal is to ensure that each call is handled efficiently by routing it to the appropriate customer service representative (CSR).

The call center has two main departments:

- Technical Support (for troubleshooting and technical issues)
- Customer Service (for account-related queries and general assistance)

Each department has its own set of CSRs, and each CSR can handle a certain number of calls at a time. The call center uses two types of queues:

- **Incoming Call Queue:** Where calls wait until they are assigned to a CSR.
- **Department Queue:** Each department has its own queue where calls wait until they are routed to an available CSR.

Technical Support has 3 CSRs, each able to handle up to 2 calls simultaneously.

Customer Service has 2 CSRs, each able to handle up to 3 calls simultaneously.

Calls arrive at the call center according to a Poisson process with an average arrival rate of 6 calls per minute. The service times for calls follow an exponential distribution:

- Technical Support: Average service time is 5 minutes.
- Customer Service: Average service time is 3 minutes.

Challenges:

- **Queue Management:** How should the call routing system manage the queues to ensure efficient handling of incoming calls and prevent long wait times?
- **System Performance:** How does the number of available CSRs and their call-handling capacity impact the average wait time for callers in each department?
- **Balancing Load:** What strategies can be used to balance the load between the two departments to prevent overload in one while the other has a low call volume?

Explanation:

To solve these challenges, we need to analyze the queuing system using concepts from queuing theory and operating systems. We can use the following methods:

Queue Management and Analysis:

Arrival Rate (λ): 6 calls per minute.

Service Rate (μ):

Technical Support: 3 CSRs × 2 calls per CSR per minute = 6 calls per minute.

Customer Service: 2 CSRs × 3 calls per CSR per minute = 6 calls per minute.

Using the M/M/c queuing model to calculate the average wait times and system utilization for each department.

Load Balancing:

Implement a dynamic call routing system that directs calls based on current wait times and the availability of CSRs.

Adjust the number of CSRs dynamically based on the call volume to optimize performance.

```
import numpy as np
import queue
import random
import threading
import time

arrival_rate = 6
technical_service_rate = 1 / 5
customer_service_rate = 1 / 3

technical_csrs = 3
customer_service_csrs = 2

incoming_queue = queue.Queue()
tech_support_queue = queue.Queue()
customer_service_queue = queue.Queue()

def call_arrival():
    # Write code here

def tech_support_service():
    # Write code here

def customer_service():
    # Write code here

def call_routing():
    # Write code here

# Driver Code
threads = []
for _ in range(technical_csrs):
    t = threading.Thread(target=tech_support_service)
    threads.append(t)
    t.start()

for _ in range(customer_service_csrs):
    t = threading.Thread(target=customer_service)
    threads.append(t)
    t.start()

t = threading.Thread(target=call_arrival)
threads.append(t)
t.start()

t = threading.Thread(target=call_routing)
threads.append(t)
```

```
t.start()

for t in threads:
    t.join()
```

13.4 The Tale of the Hospital Emergency Room (ER)

Imagine a busy hospital's Emergency Room (ER) named HealthFirst ER. The ER is designed to handle urgent medical cases and has several key components and constraints that affect how patients are treated.

The ER has two types of medical staff:

- **Doctors:** Who provide primary medical treatment?
- **Nurses:** Who assist with initial assessments and patient care?

The ER has the following setup:

- **Doctors:** There are 4 doctors, each of whom can attend to one patient at a time. The average time a doctor spends with a patient is 20 minutes.
- **Nurses:** There are 6 nurses, each of whom can handle one patient at a time. The average time a nurse spends with a patient is 10 minutes.

Patients arrive at the ER following a Poisson process with an average arrival rate of 8 patients per hour. Upon arrival, patients first see a nurse for initial assessment and then see a doctor for treatment.

The ER operates under these constraints:

- **Nurse Queue:** Patients wait here if no nurses are available.
- **Doctor Queue:** After the initial assessment, patients wait here if no doctors are available.

Challenges:

Queue Management: How should the ER manage the queues to ensure efficient handling of patients and minimize their wait times?

System Performance: How do the number of available doctors and nurses, and their patient-handling capacity, impact the average wait time for patients in the ER?

Load Balancing: How can the ER manage the flow of patients to ensure that neither the nurse nor the doctor queue becomes a bottleneck?

Explanation:

To address these challenges, we need to use concepts from queuing theory and operating systems:

Queue Management and Analysis:

Arrival Rate (λ): 8 patients per hour.

Service Rates (μ):

Nurses: 6 nurses × 1 patient per 10 minutes = 6 patients every 10 minutes or 36 patients per hour.

Doctors: 4 doctors × 1 patient per 20 minutes = 4 patients every 20 minutes or 12 patients per hour.

System Performance Analysis:

Use the M/M/c queuing model for both the nurse and doctor queues to calculate average wait times, system utilization, and other performance metrics.

Load Balancing:

Implement strategies to dynamically manage patient flow, such as adjusting the number of nurses or doctors based on patient arrival rates or queue lengths.

```
import numpy as np
import queue
import threading
import time
```

```
arrival_rate = 8  # patients per hour
nurse_service_rate = 1 / 10  # patients per minute
doctor_service_rate = 1 / 20  # patients per minute

nurse_count = 6
doctor_count = 4

nurse_queue = queue.Queue()
doctor_queue = queue.Queue()

def patient_arrival():
    # Write code here

def nurse_service():
    # Write code here

def doctor_service():
    # Write code here

# Driver Code
threads = []
for _ in range(nurse_count):
    t = threading.Thread(target=nurse_service)
    threads.append(t)
    t.start()

for _ in range(doctor_count):
    t = threading.Thread(target=doctor_service)
    threads.append(t)
    t.start()

t = threading.Thread(target=patient_arrival)
threads.append(t)
t.start()

for t in threads:
    t.join()
```

13.5 The Tale of the Online Gaming Server

Imagine a popular online multiplayer game named GameWorld. The game's server handles various player requests and needs to manage the load efficiently to ensure a smooth gaming experience. The server has two main tasks:

- **Matchmaking**: Pairing players together for matches.
- **Game State Management:** Keeping track of the state of ongoing games and ensuring that all players are synchronized.

GameWorld has the following setup:

Matchmaking System: The system pairs players based on their skill levels and game preferences. It has 3 matchmaking servers, each capable of handling up to 50 player requests per minute. The average time to match players is 1 minute.

Game State Management System: Once players are matched, their game states are managed by 2 game state servers. Each server can handle up to 30 game state updates per minute. The average time to update a game state is 2 minutes.

Players connect to the server and request matchmaking based on a Poisson process with an average arrival rate of 120 requests per minute. Once matched, these players generate game state updates also following a Poisson process with an average rate of 100 updates per minute.

Challenges:

- **Queue Management:** How should the server manage the matchmaking and game state update queues to ensure minimal wait times and efficient processing?
- **System Performance:** How do the number of matchmaking and game state servers and their capacities affect the average wait times and system utilization?
- **Load Balancing:** What strategies can be used to balance the load between matchmaking and game state management to avoid bottlenecks?

Explanation:

To address these challenges, use queuing theory and operating systems principles:

Queue Management and Analysis:

Arrival Rates (λ):

Matchmaking Requests: 120 requests per minute.

Game State Updates: 100 updates per minute.

Service Rates (μ):

Matchmaking Servers: 3 servers × 50 requests per server per minute = 150 requests per minute.

Game State Servers: 2 servers × 30 updates per server per minute = 60 updates per minute.

System Performance Analysis:

Use the M/M/c queuing model to calculate the average wait times, system utilization, and other performance metrics for both matchmaking and game state management.

Load Balancing:

Implement strategies such as dynamic load distribution and scaling resources based on current demand to balance the load effectively.

```
import numpy as np
import queue
import threading
import time

matchmaking_arrival_rate = 120
game_state_arrival_rate = 100
matchmaking_service_rate = 1 / 1
game_state_service_rate = 1 / 2

matchmaking_servers = 3
game_state_servers = 2

matchmaking_queue = queue.Queue()
game_state_queue = queue.Queue()

def player_request():
    # Write code here

def matchmaking_service():
    # Write code here

def game_state_service():
    # Write code here
# Driver Code
threads = []
for _ in range(matchmaking_servers):
    t = threading.Thread(target=matchmaking_service)
    threads.append(t)
    t.start()

for _ in range(game_state_servers):
    t = threading.Thread(target=game_state_service)
    threads.append(t)
    t.start()

t = threading.Thread(target=player_request)
threads.append(t)
t.start()

for t in threads:
    t.join()
```

V.TEXT BOOKS:

1. Abraham Silberschatz, Peter Galvin and Gagne, “Operating System Concepts”, Addison Wesley, 6th edition, 2002.
2. Harvey M.Deitel, ”Operating System”, Addison Wesley, 2nd edition, 2000.

VII. ELECTRONICS RESOURCES

1. https://cs.sdsu.edu/master-exams/operating-systems-architecture/

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

DATA STRUCTURES LABORATORY								
III Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD11	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Essentials of Problem Solving								

I. COURSE OVERVIEW:

The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSES OBJECTIVES:

The students will try to learn

I. To provide students with skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.
II. To provide knowledge of basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.
III. The fundamentals of how to store, retrieve, and process data efficiently.
IV. To provide practice by specifying and implementing these data structures and algorithms in Python.
V. Understand essential for future programming and software engineering courses.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Interpret the complexity of algorithm using the asymptotic notations.
CO 2	Select appropriate searching and sorting technique for a given problem.
CO 3	Construct programs on performing operations on linear and nonlinear data structures for organization ofa data.
CO 4	Make use of linear data structures and nonlinear data structures solving real time applications.
CO 5	Describe hashing techniques and collision resolution methods for efficiently accessing data with respectto performance.
CO 6	Compare various types of data structures; in terms of implementation, operations and performance.

IV. LIST OF EXPERIMENTS:

DATA STRUCTURES LABORATORY COURSE CONTENT

IV. COURSE CONTENT:

EXERCISES FOR DATA STRUCTURES LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Sum of last digits of two given numbers

Rohit wants to add the last digits of two given numbers. For example, If the given numbers are 267 and 154, the output should be 11.
Below is the explanation -
Last digit of the 267 is 7
Last digit of the 154 is 4
Sum of 7 and 4 = 11
Write a program to help Rohit achieve this for any given two numbers.
The prototype of the method should be -
int addLastDigits(int input1, int input2);
where input1 and input2 denote the two numbers whose last digits are to be added.
Note: The sign of the input numbers should be ignored.
if the input numbers are 267 and 154, the sum of last two digits should be 11
if the input numbers are 267 and -154, the sum of last two digits should be 11
if the input numbers are -267 and 154, the sum of last two digits should be 11
if the input numbers are -267 and -154, the sum of last two digits should be 11
Input: 267 154
Output: 11
Input: 267 -154
Output: 11
Input: -267 154
Output: 11
Input: -267 -154
Output: 11

```
import java.util.Scanner;

class AddLastDigitsFunction
{
      int addLastDigits(int n1, int n2)
      {
              # Write code here
      }

      public static void main(String args[])
      {
            AddLastDigitsFunction obj = new AddLastDigitsFunction();
            # Write code here
            System.out.println(obj.addLastDigits(n1,n2));
      }

}
```

1.2 Is N an exact multiple of M?

Write a function that accepts two parameters and finds whether the first parameter is an exact multiple of the second parameter. If the first parameter is an exact multiple of the second parameter, the function should return 2 else it should return 1.

If either of the parameters are zero, the function should return 3.

Assumption: Within the scope of this question, assume that - the first parameter can be positive, negative or zero the second parameter will always be >=0

Input: num1 = 10, num2 = 5

Output: 2

Input: num1 = -10, num2 = 5

Output: 2

Input: num1 = 0, num2 = 5

Output: 3

Input: num1 = 10, num2 = 3

Output: 1

```
public class MultipleChecker
{
    public static int checkMultiple(int num1, int num2)
    {
        # Write code here
    }

    public static void main(String[] args)
    {
        # Write code here
    }
}
```

1.3 Combine Strings

Given 2 strings, a and b, return a new string of the form short+long+short, with the shorter string on the outside and the longer string in the inside. The strings will not be the same length, but they may be empty (length 0).

If input is "hi" and "hello", then output will be "hihellohi"

Input: Enter the first string: “hi”
Enter the second string: “hello”

Output: “hihellohi”

Input: Enter the first string: “iare”
Enter the second string: “college”

Output: “iarecollegeiare”

```
public class StringCombiner
{
    public static void main(String[] args)
    {
        # Write code here
    }

    public static String combineStrings(String a, String b)
    {
        # Write code here
    }
}
```

1.4 Even or Odd

Write a function that accepts 6 input parameters. The first 5 input parameters are of type int. The sixth input parameter is of type string. If the sixth parameter contains the value "even", the function is supposed to return the count of how many of the first five input parameters are even. If the sixth parameter contains the value "odd", the function is supposed to return the count of how many of the first five input parameters are odd.

Example:

If the five input parameters are 12, 17, 19, 14, and 115, and the sixth parameter is "odd", the function must return 3, because there are three odd numbers 17, 19 and 115.

If the five input parameters are 12, 17, 19, 14, and 115, and the sixth parameter is "even", the function must return 2, because there are two even numbers 12 and 14.

Note that zero is considered an even number.

Input: num1 = 12;
num2 = 17;
num3 = 19;
num4 = 14;
num5 = 115;
type = "odd"

Output: 3

Input: num1 = 12;
num2 = 17;
num3 = 19;
num4 = 14;
num5 = 115;
type = "even"

Output: 2

```
public class NumberCounter
{
    public static int countNumbers(int num1, int num2, int num3, int num4, int num5,
String type)
   {
        # Write code here
   }

    public static void main(String[] args)
    {
        # Write code here
    }
}
```

1.5 Second last digit of a given number

Write a function that returns the second last digit of the given number. Second last digit is being referred to the digit in the tens place in the given number.

Example: if the given number is 197, the second last digit is 9.

Note 1: The second last digit should be returned as a positive number. i.e. if the given number is -197, the second last digit is 9.

Note 2: If the given number is a single digit number, then the second last digit does not exist. In such cases, the function should return -1. i.e. if the given number is 5, the second last digit should be returned as -1.

Input: 197

Output: 9

Input: 5

Output: -1

Input: -197

Output: 9

```
public class SecondLastDigit
{
    public static int getSecondLastDigit(int number)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.6 Alternate String Combiner

Given two strings, a and b, print a new string which is made of the following combination-first character of a, the first character of b, second character of a, second character of b and so on.
Any characters left, will go to the end of the result.
Hello,World
HWeolrllod
Input: "Hello,World"
Output: "HWeolrllod"
Input: "Iare,College"
Output: "ICaorlelege"

```
public class AlternateStringCombiner
{
    public static void main(String[] args)
    {
        # write code here
    }
    public static String combineStrings(String a, String b)
    {
        # write code here
    }
}
```

1.7 Padovan Sequence

The Padovan sequence is a sequence of numbers named after Richard Padovan, who attributed its discovery to Dutch architect Hans van der Laan. The sequence was described by Ian Stewart in his Scientific American column Mathematical Recreations in June 1996. The Padovan sequence is defined by the following recurrence relation:
P(n) = P(n-2) + P(n-3)
with the initial conditions P(0) = P(1) = P(2) = 1.
In this sequence, each term is the sum of the two preceding terms, similar to the Fibonacci sequence. However, the Padovan sequence has different initial conditions and exhibits different growth patterns.
The first few terms of the Padovan sequence are: 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, ...
Input: num = 10
Output: Padovan Sequence up to 10:
1 1 1 2 2 3 4 5 7 9 12
Input: num = 20
Output: Padovan Sequence up to 20:
1 1 1 2 2 3 4 5 7 9 12 16 21 28 37 49 65 86 114 151 200

```
public class PadovanSequence
```

```
{
    public static int padovan(int n)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.8 Leaders in an array

Given an array arr of n positive integers, your task is to find all the leaders in the array. An element of the array is considered a leader if it is greater than all the elements on its right side or if it is equal to the maximum element on its right side. The rightmost element is always a leader.

Input: n = 6, arr[] = (16, 17, 4, 3, 5, 2}
Output: 17 5 2
Input: n = 5, arr[] = {10, 4, 2, 4, 1}
Output: 10 4 4 1
Input: n = 4, arr[] = {5, 10, 20, 40}
Output: 40
Input: n = 4, arr[] = {30, 10, 10, 5}
Output: 30 10 10 5

```
import java.util.ArrayList;
import java.util.List;
public class ArrayLeaders
{
    public static List<Integer> findArrayLeaders(int[] arr)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.9 Find the Value of a Number Raised to its Reverse

Given a number N and its reverse R. The task is to find the number obtained when the number is raised to the power of its own reverse

Input : N = 2, R = 2
Output: 4
Explanation: Number 2 raised to the power of its reverse 2 gives 4 which gives 4 as a result after performing modulo 10^9+7

Input: N = 57, R = 75
Output: 262042770
Explanation: 57^{75} modulo 10^9+7 gives us the result as 262042770

```
public class NumberPower
{
    public static long powerOfReverse(int N, int R)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.10 Mean of Array using Recursion

Find the mean of the elements of the array.
Mean = (Sum of elements of the Array) / (Total no of elements in Array)
Input: 1 2 3 4 5
Output: 3.0
Input: 1 2 3
Output: 2.0
To find the mean using recursion assume that the problem is already solved for N-1 i.e. you have to find for n
Sum of first N-1 elements = (Mean of N-1 elements) * (N-1)
Mean of N elements = (Sum of first N-1 elements + N-th elements) / (N)

```
public class ArrayMean
{
    public static double findArrayMean(int[] arr)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

Try:

1. **Kth Smallest Element:** Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth smallest element in the given array. It is given that all array elements are distinct.
Note: l and r denotes the starting and ending index of the array.
Input: n = 6, arr[] = {7, 10, 4, 3, 20, 15}, k = 3, l = 0, r = 5
Output: 7
Explanation: 3rd smallest element in the given array is 7.
Input: n = 5, arr[] = {7, 10, 4, 20, 15}, k = 4, l=0 r=4
Output: 15

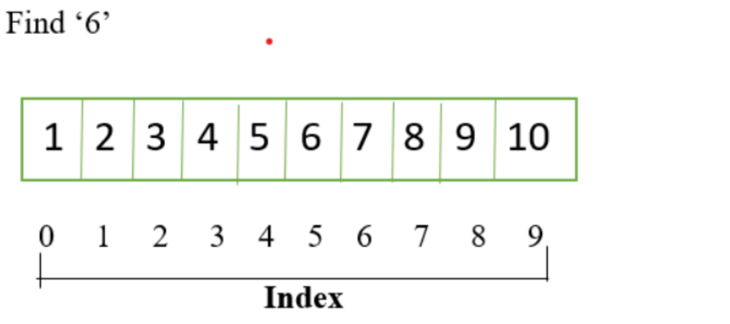
Explanation: 4th smallest element in the given array is 15.
Your task is to complete the function **kthSmallest()** which takes the array arr[], integers l and r denoting the starting and ending index of the array and an integer k as input and returns the kth smallest element.
2. **Count pairs with given sum:** Given an array of N integers, and an integer K, find the number of pairs of elements in the array whose sum is equal to K. Your task is to complete the function **getPairsCount()** which takes arr[], n and k as input parameters and returns the number of pairs that have sum K.
Input: N = 4, K = 6, arr[] = {1, 5, 7, 1}
Output: 2
Explanation: arr[0] + arr[1] = 1 + 5 = 6 and arr[1] + arr[3] = 5 + 1 = 6.
Input: N = 4, K = 2, arr[] = {1, 1, 1, 1}
Output: 6

Explanation: Each 1 will produce sum 2 with any 1.

2. Searching

2.1 Linear / Sequential Search

Linear search is defined as the searching algorithm where the list or data set is traversed from one end to find the desired value. Given an array arr[] of n elements, write a recursive function to search a given element x in arr[].



Note : We find '6' at index '5' through linear search

Linear search procedure:

1. Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
2. If x matches with an element, return the index.
3. If x doesn't match with any of the elements, return -1.

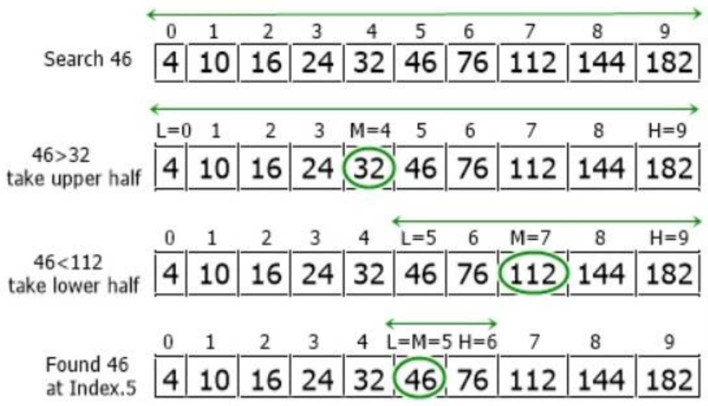
Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 110;
Output: 6
Element x is present at index 6

Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 175;
Output: -1
Element x is not present in arr[].

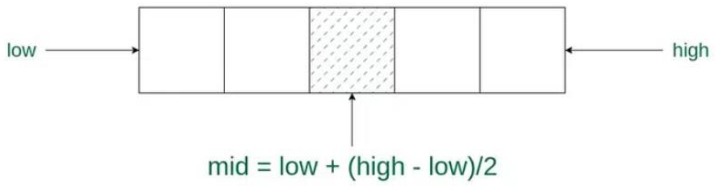
```
public class RecursiveLinearSearch
{
    public static int recursiveLinearSearch(int[] arr, int key, int index)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

2.2 Binary Search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).



Conditions for Binary Search algorithm:
1. The data structure must be sorted.
2. Access to any element of the data structure takes constant time.



Binary Search Procedure:
1. Divide the search space into two halves by finding the middle index "mid".
2. Compare the middle element of the search space with the key.
3. If the key is found at middle element, the process is terminated.
4. If the key is not found at middle element, choose which half will be used as the next search space.
   a. If the key is smaller than the middle element, then the left side is used for next search.
   b. If the key is larger than the middle element, then the right side is used for next search.
5. This process is continued until the key is found or the total search space is exhausted.

Input: arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
Output: target = 23
Element 23 is present at index 5

```
public class RecursiveBinarySearch
{
    public static int recursiveBinarySearch(int[] arr, int key, int left, int right)
    {
         # write code here
    }

    public static void main(String[] args)
    {
         # write code here
    }
}
```

2.3 Uniform Binary Search

Uniform Binary Search is an optimization of Binary Search algorithm when many searches are made on same array or many arrays of same size. In normal binary search, we do arithmetic operations to find the mid points. Here we precompute mid points and fills them in lookup table. The array look-up generally works faster than arithmetic done (addition and shift) to find the mid-point.

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=3
Output: Position of 3 in array = 2

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=7
Output: Position of 7 in array = 5

The algorithm is very similar to Binary Search algorithm, the only difference is a lookup table is created for an array and the lookup table is used to modify the index of the pointer in the array which makes the search faster. Instead of maintaining lower and upper bound the algorithm maintains an index and the index is modified using the lookup table.

```
public class RecursiveUniformBinarySearch
{
    public  static  int  recursiveUniformBinarySearch(int[]  arr,  int  key,  int[]
lookupTable, int left, int right)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

2.4 Interpolation Search

Interpolation search works better than Binary Search for a Sorted and Uniformly Distributed array. Binary search goes to the middle element to check irrespective of search-key. On the other hand, Interpolation search may go to different locations according to search-key. If the value of the search-key is close to the last element, Interpolation Search is likely to start search toward the end side. Interpolation search is more efficient than binary search when the elements in the list are uniformly distributed, while binary search is more efficient when the elements in the list are not uniformly distributed.

Interpolation search can take longer to implement than binary search, as it requires the use of additional calculations to estimate the position of the target element.

Input: arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Output: target = 5

```
public class InterpolationSearch
{
    public static int interpolationSearch(int[] arr, int key)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

2.5 Fibonacci Search

Given a sorted array arr[] of size n and an element x to be searched in it. Return index of x if it is present in array else return -1.

Input: arr[] = {2, 3, 4, 10, 40}, x = 10
Output: 3
Element x is present at index 3.

Input: arr[] = {2, 3, 4, 10, 40}, x = 11
Output: -1
Element x is not present.

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Fibonacci Search Procedure:
Let the searched element be x. The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of the given array. Let the found Fibonacci number be fib (m'th Fibonacci number). We use (m-2)'th Fibonacci number as the index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Let arr[0..n-1] be the input array and the element to be searched be x.

1. Find the smallest Fibonacci number greater than or equal to n. Let this number be fibM [m'th Fibonacci number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].
2. While the array has elements to be inspected:
   i. Compare x with the last element of the range covered by fibMm2
   ii. If x matches, return index
   iii. Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
   iv. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array.
3. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

```
public class FibonacciSearch
{
    public static int fibonacciSearch(int[] arr, int key)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

3. Sorting

3.1 Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

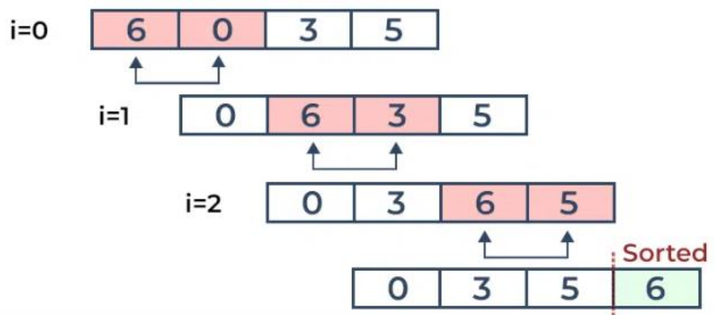
Bubble Sort Procedure:

1. Traverse from left and compare adjacent elements and the higher one is placed at right side.
2. In this way, the largest element is moved to the rightmost end at first.
3. This process is then continued to find the second largest and place it and so on until the data is sorted.

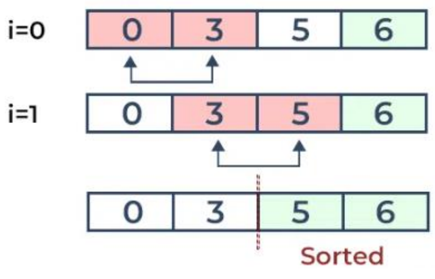
Input: arr = [6, 3, 0, 5]

Output:

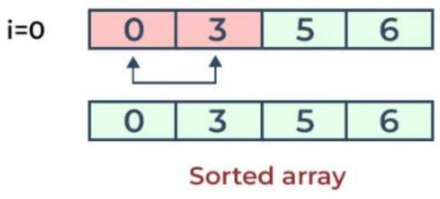
First Pass:



Second Pass:



Third Pass:



```
import java.util.Scanner;

class BubbleSortExample
{
    public static void main(String[] args)
    {
          # write code here

    }
    public static void bubbleSort(int[] arr)
    {
         # write code here

    }
}
```

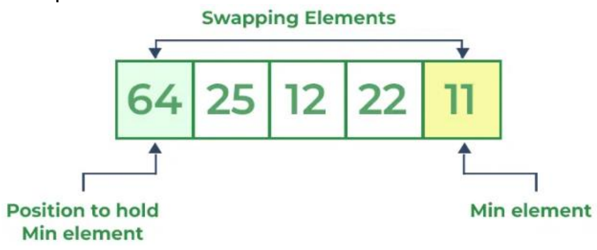
3.2 Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

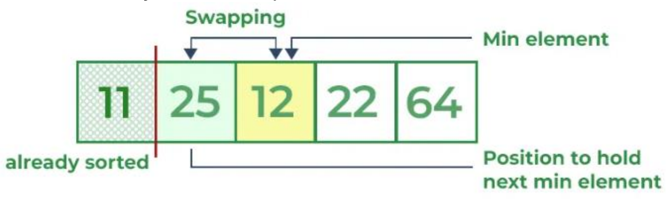
Input: arr = [64, 25, 12, 22, 11]

Output: arr = [11, 12, 22, 25, 64]

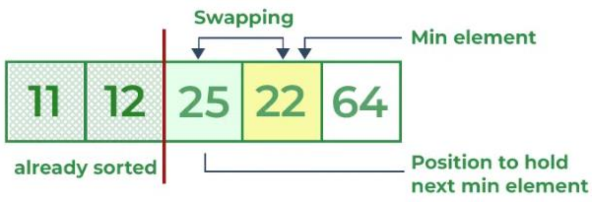
First Pass: For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value. Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.



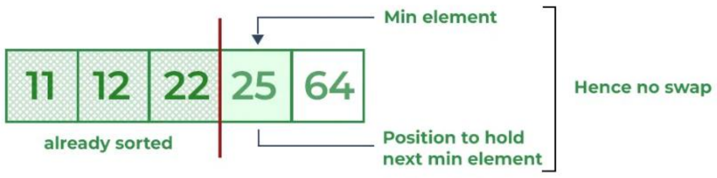
Second Pass: For the second position, where 25 is present, again traverse the rest of the array in a sequential manner. After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



Third Pass: Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array. While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



Fourth Pass: Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array. As 25 is the 4th lowest value hence, it will place at the fourth position.



Fifth Pass: At last the largest value present in the array automatically get placed at the last position in the array. The resulted array is the sorted array.

11
12
22
25
64

Sorted array

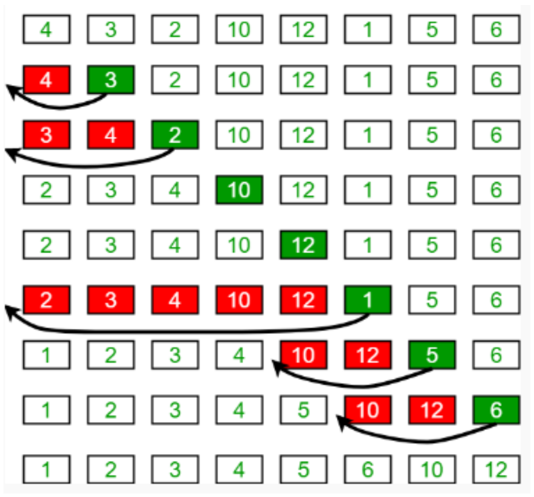
```
import java.util.Scanner;

class SelectionSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void selectionSort(int[] arr)
    {
        # write code here
    }
}
```

3.3 Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion Sort Procedure:

1. To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before.
2. Move the greater elements one position up to make space for the swapped element.



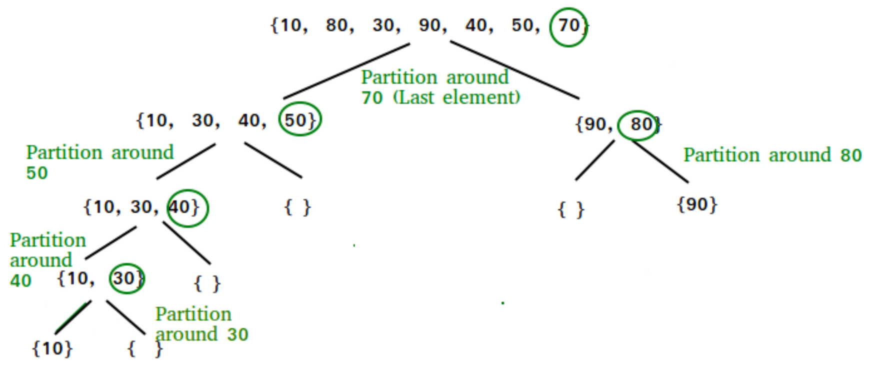
Input: arr = [4, 3, 2, 10, 12, 1, 5, 6]
Output: arr = [1, 2, 3, 4, 5, 6, 10, 12]

```
import java.util.Scanner;

class InsertionSortExample
{
    public static void main(String[] args)
    {
       # write code here
    }
    public static void insertionSort(int[] arr)
    {
        # write code here
    }
}
```

4. Divide and Conquer

4.1 Quick Sort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



The quick sort method can be summarized in three steps:
1. **Pick:** Select a pivot element.
2. **Divide:** Split the problem set, move smaller parts to the left of the pivot and larger items to the right.
3. **Repeat and combine:** Repeat the steps and combine the arrays that have previously been sorted.

Algorithm for Quick Sort Function:

```
//start –> Starting index,  end --> Ending index
Quicksort(array, start, end)
{
        if (start < end)
        {
                pIndex = Partition(A, start, end)
                Quicksort(A,start,pIndex-1)
                Quicksort(A,pIndex+1, end)
        }
}
```

Algorithm for Partition Function:

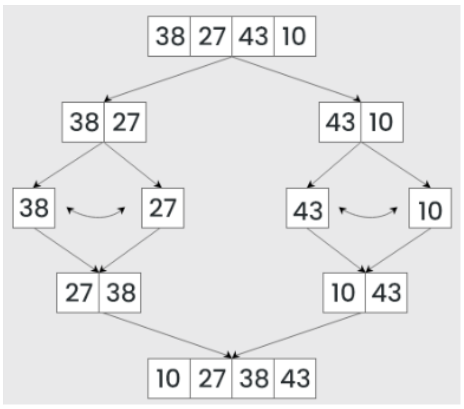
```
partition (array, start, end)
{
        // Setting rightmost Index as pivot
        pivot = arr[end];

        i = (start - 1)  // Index of smaller element and indicates the
            // right position of pivot found so far
        for (j = start; j <= end- 1; j++)
        {
                // If current element is smaller than the pivot
                if (arr[j] < pivot)
                {
                        i++;    // increment index of smaller element
                        swap arr[i] and arr[j]
                }
        }
        swap arr[i + 1] and arr[end])
        return (i + 1)
}
```

Input: arr = [10, 80, 30, 90, 40, 50, 70]
Output: arr = [10, 30, 40, 50, 70, 80, 90]

```
import java.util.Scanner;

class QuickSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void quickSort(int[] arr, int low, int high)
    {
        # write code here
    }

    public static int partition(int[] arr, int low, int high)
    {
        # write code here
    }
}
```

4.2 Merge Sort

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



Input: arr = [12, 11, 13, 5, 6, 7]
Output: arr = [5, 6, 7, 11, 12, 13]

```
import java.util.Scanner;

class MergeSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void mergeSort(int[] arr, int low, int high)
    {
        # write code here
    }

    public static void merge(int[] arr, int low, int mid, int high)
    {
        # write code here
    }
}
```

4.3 Heap Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Heap Sort Procedure:

First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
  - Swap the root element of the heap (which is the largest element) with the last element of the heap.
  - Remove the last element of the heap (which is now in the correct position).
  - Heapify the remaining elements of the heap.
- The sorted array is obtained by reversing the order of the elements in the input array.

Input: arr = [12, 11, 13, 5, 6, 7]
Output: Sorted array is 5 6 7 11 12 13

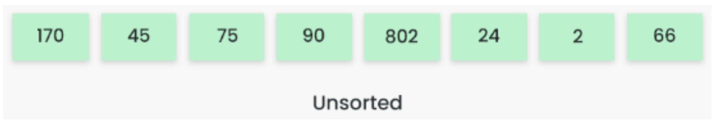
```
import java.util.Scanner;

class HeapSortExample
{
    public static void main(String[] args)
    {
                # write code here
    }

    public static void heapSort(int[] arr)
    {
                # write code here
    }

    public static void heapify(int[] arr, int n, int i)
    {
        # write code here
    }
}
```

4.4 Radix Sort

Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys. Rather than comparing elements directly, Radix Sort distributes the elements into buckets based on each digit's value. By repeatedly sorting the elements by their significant digits, from the least significant to the most significant, Radix Sort achieves the final sorted order.

Radix Sort Procedure:
The key idea behind Radix Sort is to exploit the concept of place value.
1. It assumes that sorting numbers digit by digit will eventually result in a fully sorted list.
2. Radix Sort can be performed using different variations, such as Least Significant Digit (LSD) Radix Sort or Most Significant Digit (MSD) Radix Sort.

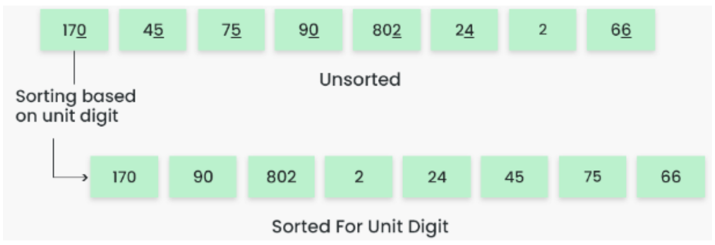
To perform radix sort on the array [170, 45, 75, 90, 802, 24, 2, 66], we follow these steps:



Step 1: Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.

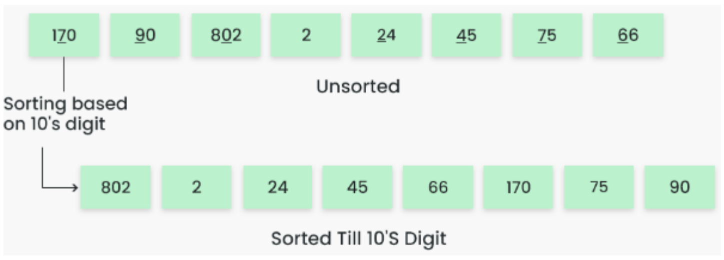
Step 2: Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place.

Sorting based on the unit place:
Perform counting sort on the array based on the unit place digits.
The sorted array based on the unit place is [170, 90, 802, 2, 24, 45, 75, 66]



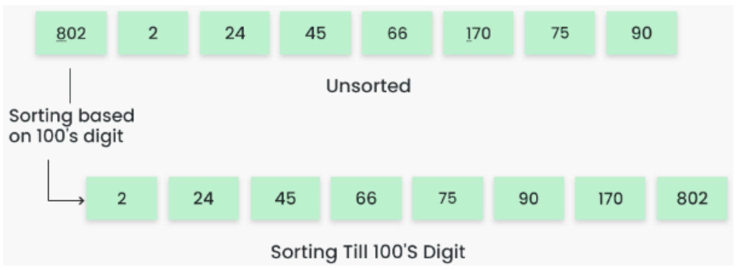
Step 3: Sort the elements based on the tens place digits.

Sorting based on the tens place:
Perform counting sort on the array based on the tens place digits.
The sorted array based on the tens place is [802, 2, 24, 45, 66, 170, 75, 90]



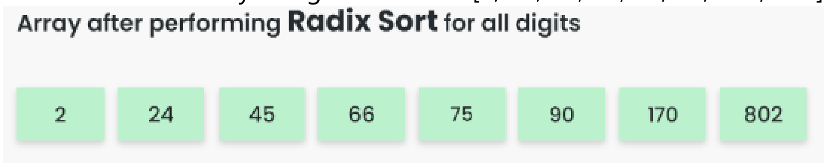
Step 4: Sort the elements based on the hundreds place digits.

Sorting based on the hundreds place:
Perform counting sort on the array based on the hundreds place digits.
The sorted array based on the hundreds place is [2, 24, 45, 66, 75, 90, 170, 802]



Step 5: The array is now sorted in ascending order.

The final sorted array using radix sort is [2, 24, 45, 66, 75, 90, 170, 802]



```
import java.util.Arrays;

class RadixSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void radixSort(int[] arr)
    {
        # write code here
    }

    public static int getMax(int[] arr)
    {
        # write code here
    }

    public static void countSort(int[] arr, int exp)
    {
        # write code here
    }
}
```

4.5 Shell Sort

Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

Shell Sort Procedure:

1. Initialize the value of gap size h
2. Divide the list into smaller sub-part. Each must have equal intervals to h
3. Sort these sub-lists using insertion sort
4. Repeat this step 1 until the list is sorted.
5. Print a sorted list.

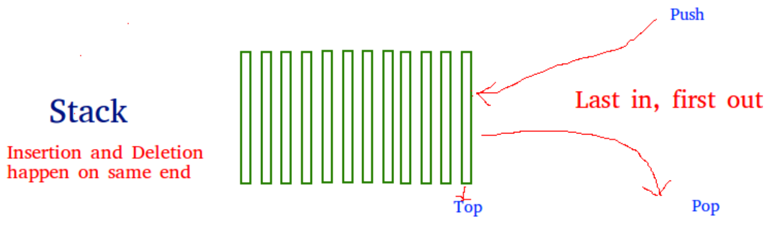
```
Procedure Shell_Sort(Array, N)
   While Gap < Length(Array) /3 :
                    Gap = ( Interval * 3 ) + 1
   End While Loop
   While Gap > 0 :
       For (Outer = Gap; Outer < Length(Array); Outer++):
             Insertion_Value = Array[Outer]
                    Inner = Outer;
             While Inner > Gap-1 And Array[Inner – Gap] >= Insertion_Value:
                    Array[Inner] = Array[Inner – Gap]
                    Inner = Inner – Gap
              End While Loop
                  Array[Inner] = Insertion_Value
       End For Loop
       Gap = (Gap -1) /3;
   End While Loop
End Shell_Sort
```

```
import java.util.Scanner;

class ShellSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void shellSort(int[] arr)
    {
        # write code here
    }
}
```

5. Stack

5.1 Implementation of Stack

A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty
- **size()** – Returns the size of the stack
- **top() / peek()** – Returns a reference to the topmost element of the stack
- **push(a)** – Inserts the element 'a' at the top of the stack
- **pop()** – Deletes the topmost element of the stack

```
class Stack
{
    private int maxSize;
    private int top;
    private int[] stackArray;

    public Stack(int size)
    {
        # write code here
    }

    public void push(int value)
    {
        # write code here
    }

    public int pop()
    {
        # write code here
    }

    public int peek()
    {
        # write code here
    }

    public boolean isEmpty()
    {
        # write code here
    }

    public boolean isFull()
    {
        # write code here
    }
}

class StackExample
{
    public static void main(String[] args)
    {
        Stack stack = new Stack(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.pop();
        stack.peek();
        stack.push(40);
        stack.push(50);
        stack.push(60);
    }
}
```

5.2 Balanced Parenthesis Checking

Given an expression string, write a java program to find whether a given string has balanced parentheses or not.

Input: "{(a+b)*(c-d)}"

Output: true

Input: "{(a+b)*[c-d)}"

Output: false

One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return true otherwise, false

```
import java.util.Stack;

class BalancedParenthesisChecker
{
    public static boolean isBalanced(String expression)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        String expression1 = "{(a+b)*(c-d)}";
        String expression2 = "{(a+b)*[c-d)}";

        # write code here
    }
}
```

5.3 Evaluation of Postfix Expression

Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.

Input: str = "2 3 1 * + 9 -"

Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = "100 200 + 2 / 5 * 7 +"

Output: 757

Procedure for evaluation postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

```
import java.util.Stack;

class PostfixEvaluator
{
    public static int evaluatePostfix(String expression)
    {
        # write code here
    }

    public static int performOperation(char operator, int operand1, int operand2)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

5.4 Infix to Postfix Expression Conversion

For a given Infix expression, convert it into Postfix form.
Infix expression: The expression of the form "a operator b" (a + b) i.e., when an operator is in-between every pair of operands.
Postfix expression: The expression of the form "a b operator" (ab+) i.e., When every pair of operands is followed by an operator.

Infix to postfix expression conversion procedure:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following
    - If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack [or the stack is empty or the stack contains a '('], then push it in the stack. ['^' operator is right associative and other operators like '+','–','*' and '/' are left-associative].
        - Check especially for a condition when the operator at the top of the stack and the scanned operator both are '^'. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack.
        - In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.
    - Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator.
        - After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is a '(', push it to the stack.
5. If the scanned character is a ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-5 until the infix expression is scanned.
7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.
8. Finally, print the postfix expression.

Input: A + B * C + D
Output: A B C * + D +

Input: ((A + B) – C * (D / E)) + F
Output: A B + C D E / * - F +

```
import java.util.Stack;

class Conversion
      {

        # Write Code Here

      }
```

5.5 Reverse a Stack

The stack is a linear data structure which works on the LIFO concept. LIFO stands for last in first out. In the stack, the insertion and deletion are possible at one end the end is called the top of the stack. Define two recursive functions BottomInsertion() and Reverse() to reverse a stack using Python. Define some basic function of the stack like push(), pop(), show(), empty(), for basic operation like respectively append an item in stack, remove an item in stack, display the stack, check the given stack is empty or not.

BottomInsertion(): this method append element at the bottom of the stack and BottomInsertion accept two values as an argument first is stack and the second is elements, this is a recursive method.

Reverse(): the method is reverse elements of the stack, this method accept stack as an argument Reverse() is also a Recursive() function. Reverse() is invoked BottomInsertion() method for completing the reverse operation on the stack.

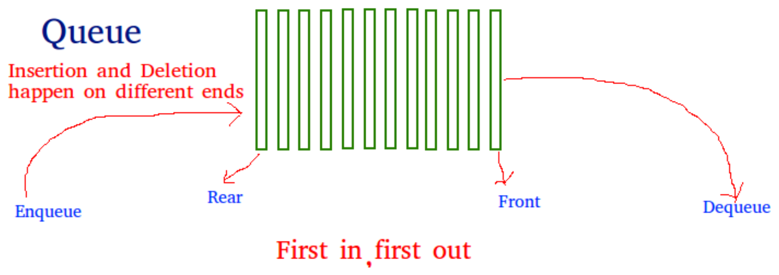
Input: Elements = [1, 2, 3, 4, 5]
Output: Original Stack
5
4
3
2
1
Stack after Reversing
1
2
3
4
5

```
import java.util.Stack;
class StackClass {
        # Write Code Here
    }
```

6. Queue

6.1 Linear Queue

Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



```
import java.util.Scanner;

public class LinearQueue
       {
       # Write Code Here
       }
    public static boolean isEmpty() {
        return front == rear;
    }
    public static boolean isFull() {
        return rear == MAX;
    }
    public static void enqueue(int item)
       {
       # Write Code Here
       }

    public static void dequeue()
       {
       # Write Code Here
       }
public static void display()
       {
        # Write Code Here
       }
    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

6.2 Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Input:
["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]
Output:
[null, null, null, 2, 2, false]

```
import java.util.LinkedList;
import java.util.Queue;

class MyStack
          {
            # Write Code Here
          }
    public void push(int x)
       {
        # Write Code Here
       }
    }

    public int pop()
       {
        return queue.remove();
       }

    public int top() {
        return queue.peek();
    }

    public boolean empty() {
        return queue.isEmpty();
    }
    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

6.3 Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Input:
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
Output:
[null, null, null, 1, 1, false]

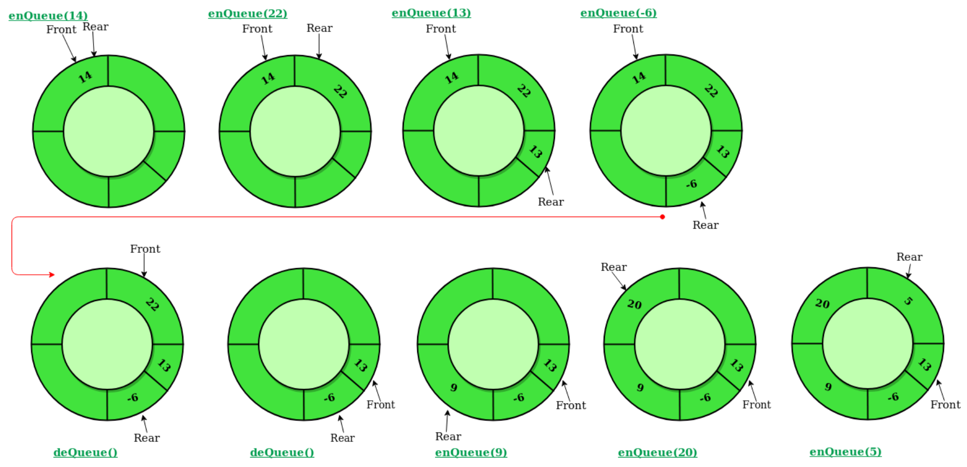
```
import java.util.Stack;

class MyQueue {

    private Stack<Integer> stack1;
    private Stack<Integer> stack2;
    public MyQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }
    public void push(int x) {
        stack1.push(x);
    }
    public int pop()
      {
        # Write Code Here
      }
    public int peek()
      {
      # Write Code Here
      }
    public boolean empty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }
    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

6.4 Circular Queue

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
        - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.



Implement Circular Queue using Array:

5. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
6. Initialize two variables front and rear to -1.
7. **Enqueue:** To enqueue an element **x** into the queue, do the following:
   - Increment rear by 1.
     - If **rear** is equal to n, set **rear** to 0.
   - If **front** is -1, set **front** to 0.
   - Set queue[rear] to x.
8. **Dequeue:** To dequeue an element from the queue, do the following:
   - Check if the queue is empty by checking if **front** is -1.
     - If it is, return an error message indicating that the queue is empty.
   - Set **x** to queue [front].
   - If **front** is equal to **rear**, set **front** and **rear** to -1.
   - Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
   - Return x.

```
class CircularQueue {

    private int size;
    private int front, rear;
    private int[] queue;

    public CircularQueue(int size) {
        this.size = size;
        this.queue = new int[size];
        this.front = this.rear = -1;
    }


    public void enqueue(int data)
      {
      # Write Code Here
      }
    public int dequeue()
      {
        # Write Code Here
      }
    public void display()
      {
      # Write Code Here
      }
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

6.5 Deque (Doubly Ended Queue)

In a Deque (Doubly Ended Queue), one can perform insert (append) and delete (pop) operations from both the ends of the container. There are two types of Deque:

1. **Input Restricted Deque:** Input is limited at one end while deletion is permitted at both ends.
2. **Output Restricted Deque:** Output is limited at one end but insertion is permitted at both ends.

Operations on Deque:

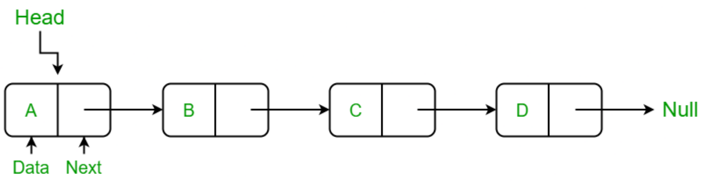
1. **append():** This function is used to insert the value in its argument to the right end of the deque.
2. **appendleft():** This function is used to insert the value in its argument to the left end of the deque.
3. **pop():** This function is used to delete an argument from the right end of the deque.
4. **popleft():** This function is used to delete an argument from the left end of the deque.
5. **index(ele, beg, end):** This function returns the first index of the value mentioned in arguments, starting searching from beg till end index.
6. **insert(i, a):** This function inserts the value mentioned in arguments(a) at index(i) specified in arguments.
7. **remove():** This function removes the first occurrence of the value mentioned in arguments.
8. **count():** This function counts the number of occurrences of value mentioned in arguments.
9. **len(dequeue):** Return the current size of the dequeue.
10. **Deque[0]:** We can access the front element of the deque using indexing with de[0].
11. **Deque[-1]:** We can access the back element of the deque using indexing with de[-1].
12. **extend(iterable):** This function is used to add multiple values at the right end of the deque. The argument passed is iterable.
13. **extendleft(iterable):** This function is used to add multiple values at the left end of the deque. The argument passed is iterable. Order is reversed as a result of left appends.
14. **reverse():** This function is used to reverse the order of deque elements.
15. **rotate():** This function rotates the deque by the number specified in arguments. If the number specified is negative, rotation occurs to the left. Else rotation is to right.

```
import java.util.ArrayDeque;
import java.util.Deque;

public class DequeOperations
{
    # Write Code Here
}
```

7. Linked List

7.1 Singly Linked List

A singly linked list is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.



Creating a linked list involves the following operations:

1. Creating a Node class:
2. Insertion at beginning:
3. Insertion at end
4. Insertion at middle
5. Update the node
6. Deletion at beginning
7. Deletion at end
8. Deletion at middle
9. Remove last node
10. Linked list traversal
11. Get length

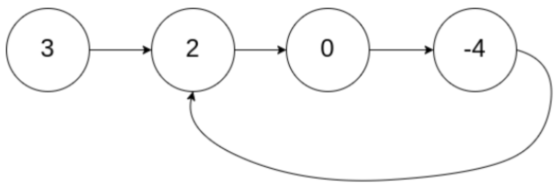
```
class Node {
    String data;
    Node next;

    Node(String data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList
      {
      # Write Code Here
      }

    public void insertAtEnd(String data)
      {
        # Write Code Here
      }

    public void updateNode(String val, int index)
      {
      # Write Code Here
      }

    public void remove_first_node() {
        # Write Code Here
    }

    public void remove_last_node()
      {
       # Write Code Here
      }

    public void remove_at_index(int index)

       {
      # Write Code Here
       }

    public void remove_node(String data)
      {
        # Write Code Here
       }

    public int sizeOfLL()
      {
    # Write Code Here
      }

    public void printLL()
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

7.2 Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

Note that pos is not passed as a parameter.

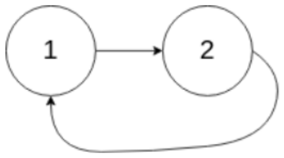
Return true if there is a cycle in the linked list. Otherwise, return false.



Input: head = [3, 2, 0, -4], pos = 1

Output: true

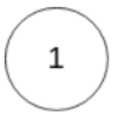
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).



Input: head = [1, 2], pos = 0

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

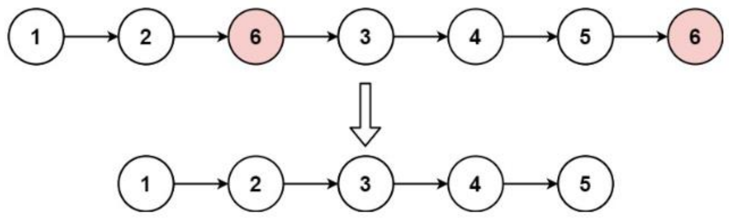


Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.

```
class ListNode
      {
    # Write Code Here
      }

public class Solution
      {
       # Write Code Here
      }
}
```

7.3 Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.



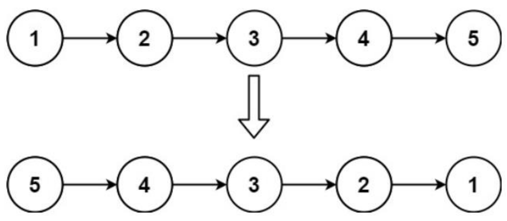
Input: head = [1, 2, 6, 3, 4, 5, 6], val = 6
Output: [1, 2, 3, 4, 5]
Input: head = [ ], val = 1
Output: [ ]
Input: head = [7, 7, 7, 7], val = 7
Output: [ ]

```
class ListNode {
    # Write Code Here
    }
}
public class Solution {
    public boolean hasCycle(ListNode head)
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

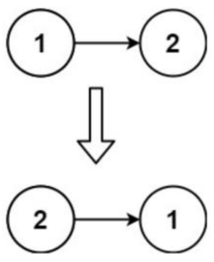
7.4 Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Input: head = [1, 2, 3, 4, 5]
Output: [5, 4, 3, 2, 1]



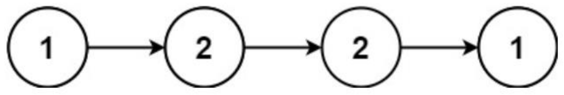
Input: head = [1, 2]
Output: [2, 1]



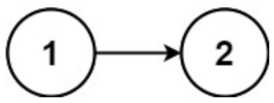
```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class Solution {
    public ListNode reverseList(ListNode head)
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
```

7.5 Palindrome Linked List

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.



Input: head = [1, 2, 2, 1]
Output: true



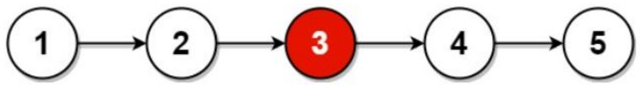
Input: head = [1, 2]
Output: false

```
class ListNode
```

```
        {
        # Write Code Here
        }
public class Solution {
    public boolean isPalindrome(ListNode head)
        {
          # Write Code Here
        }

    public static void main(String[] args)
        {
        # Write Code Here
        }
}
```

7.6 Middle of the Linked List

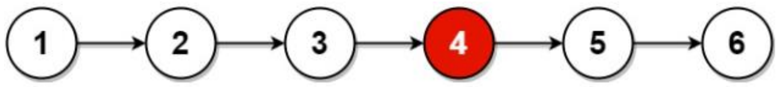
Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.



Input: head = [1, 2, 3, 4, 5]

Output: [3, 4, 5]

Explanation: The middle node of the list is node 3.



Input: head = [1, 2, 3, 4, 5, 6]

Output: [4, 5, 6]

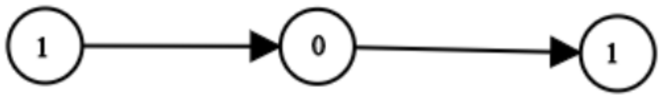
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

```
class ListNode
        {
        # Write Code Here
        }
}
public class Solution
    {
        # Write Code Here
    }

    public static void main(String[] args) {
         # Write Code Here
    }
}
```

7.7 Convert Binary Number in a Linked List to Integer

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list. The most significant bit is at the head of the linked list.



Input: head = [1, 0, 1]

Output: 5

Explanation: (101) in base 2 = (5) in base 10

Input: head = [0]

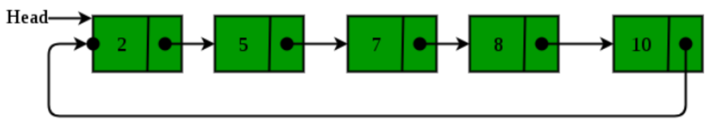
Output: 0

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class Solution
       {
          # Write Code Here
       }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

8. Circular Single Linked List and Doubly Linked List

8.1 Circular Linked List

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



Operations on the circular linked list:

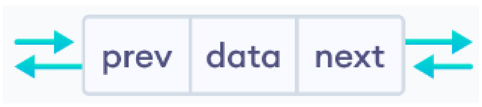
1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

```
import java.util.ArrayList;
public class Main{
	static class Node{
		int data;
		Node next;
		Node(int data){
			this.data = data;
			this.next = null;
		}
	}
	static class CircularLinkedList
	{
	# Write Code Here
	}
	Node addAfter(int data, int item)
	{
	# Write Code Here
	}
void deleteNode(Node last, int key)
	{
	# Write Code Here
	}
				System.
```

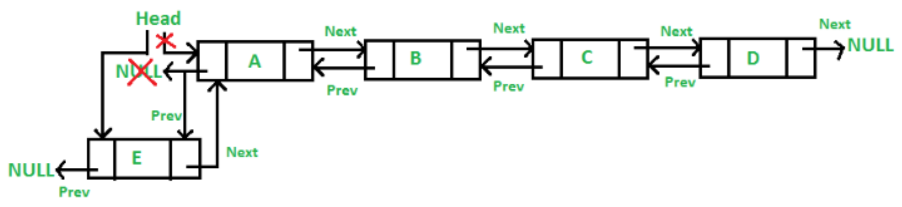
8.2 Doubly Linked List

The A doubly linked list is a type of linked list in which each node consists of 3 components:

1. *prev - address of the previous node
2. data - data item
3. *next - address of next node.



Double Linked List Node



Operations on the Double Linked List:

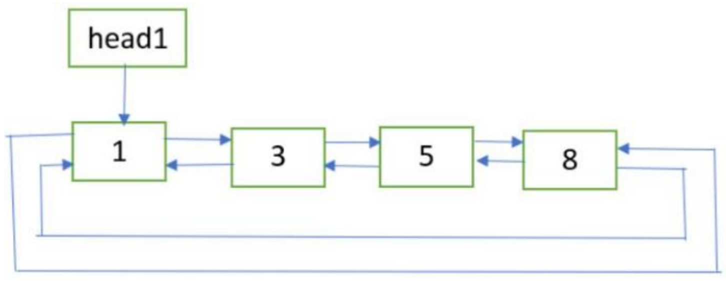
1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

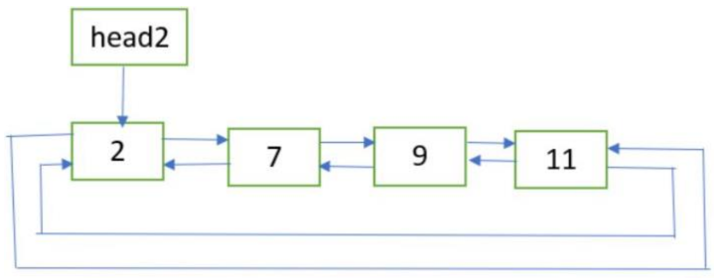
```
import java.util.Scanner;

class Node {
    int data;
    Node next;
    Node prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

class DLinkedList {
    Node head;
    int ctr;

    DLinkedList() {
        this.head = null;
        this.ctr = 0;
    }

    void insertBeg(int data)
      {
      # Write Code Here
      }
    void insertEnd(int data)
      {
      # Write Code Here
      }

    void deleteBeg()
      {
      # Write Code Here
      }
    void deleteEnd()
      {
        # Write Code Here
      }

    void insertPos(int pos, int data)
      {
      # Write Code Here
      }

    void deletePos(int pos)
      {
        # Write Code Here
      }

    void traverseF()
      {
        # Write Code Here
      }

    void traverseR()
      {
       # Write Code Here
      }

public class Main {
    public static void main(String[] args)                                   {
       # Write Code Here
      }
}
```

8.3 Sorted Merge of Two Sorted Doubly Circular Linked Lists

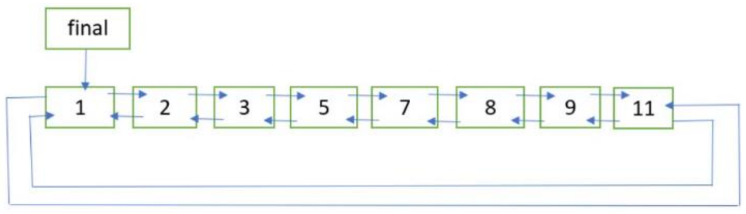
Given two sorted Doubly circular Linked List containing n1 and n2 nodes respectively. The problem is to merge the two lists such that resultant list is also in sorted order.

Input: List 1 and List 2





Output: Merged List



Procedure for Merging Doubly Linked List:

1. If head1 == NULL, return head2.
2. If head2 == NULL, return head1.
3. Let **last1** and **last2** be the last nodes of the two lists respectively. They can be obtained with the help of the previous links of the first nodes.
4. Get pointer to the node which will be the last node of the final list. If last1.data < last2.data, then **last_node** = last2, Else **last_node** = last1.
5. Update last1.next = last2.next = NULL.
6. Now merge the two lists as two sorted doubly linked list are being merged. Refer **merge** procedure of this post. Let the first node of the final list be **finalHead**.
7. Update finalHead.prev = last_node and last_node.next = finalHead.
8. Return **finalHead**.

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class SortedMergeDoublyCircularLinkedList
      {
      # Write Code Here
      }

    static Node mergeUtil(Node head1, Node head2)
      {
      # Write Code Here
      }

    static void printList(Node head)
      {
      # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

8.4 Delete all occurrences of a given key in a Doubly Linked List

Given a doubly linked list and a key x. The problem is to delete all occurrences of the given key x from the doubly linked list.

Input: 2 <-> 2 <-> 10 <-> 8 <-> 4 <-> 2 <-> 5 <-> 2

        x = 2
Output: 10 <-> 8 <-> 4 <-> 5

Algorithm:
delAllOccurOfGivenKey (head_ref, x)

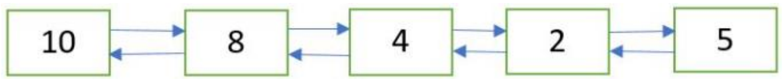
```
if head_ref == NULL
   return
Initialize **current** = head_ref
Declare **next**
while current != NULL
    if current->data == x
       next = current->next
       **deleteNode(head_ref, current)**
       current = next
    else
       current = current->next
```

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class DeleteOccurrenceInDoublyLinkedList
      {
       # Write Code Here
      }

    static Node deleteAllOccurOfX(Node head, int x)
      {
      # Write Code Here
      }

    static void printList(Node head)
      {
       # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

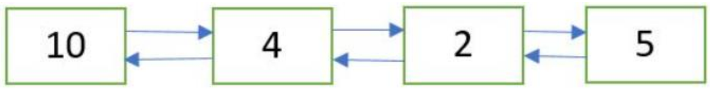
8.5 Delete a Doubly Linked List Node at a Given Position

Given a doubly linked list and a position n. The task is to delete the node at the given position n from the beginning.

Input: Initial doubly linked list



Output: Doubly Linked List after deletion of node at position n = 2



Procedure:

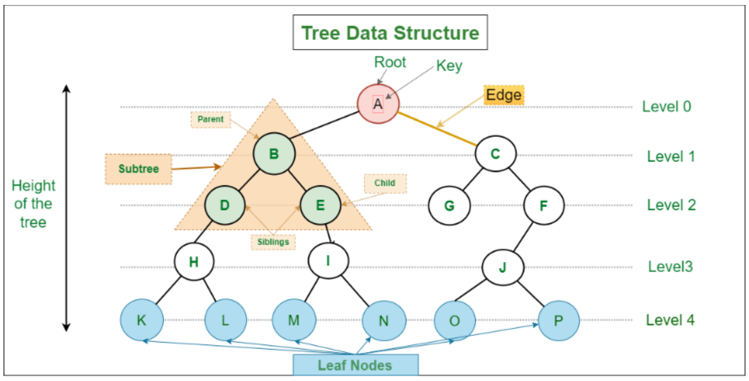
1. Get the pointer to the node at position n by traversing the doubly linked list up to the nth node from the beginning.
2. Delete the node using the pointer obtained in Step 1.

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class DeleteNodeAtGivenPosition
     {
       # Write Code Here
     }

    static Node deleteNode(Node head, Node del)
     {
        # Write Code Here
     }

    static Node deleteNodeAtGivenPos(Node head, int n)
     {
        # Write Code Here
     }
    static void printList(Node head)
     {
      # Write Code Here
     }
}
```

9. Trees

9.1 Tree Creation and Basic Tree Terminologies

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.



Basic Terminologies in Tree:

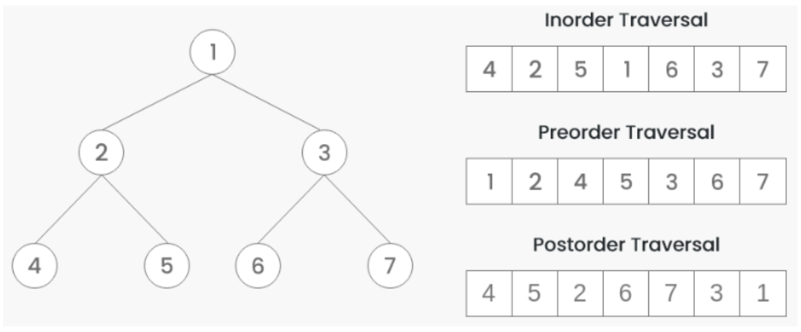
1. **Parent Node:** The node which is a predecessor of a node is called the parent node of that node. {B} is the parent node of {D, E}.
2. **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: {D, E} are the child nodes of {B}.
3. **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
4. **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. {K, L, M, N, O, P} are the leaf nodes of the tree.
5. **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {A, B} are the ancestor nodes of the node {E}
6. **Descendant:** Any successor node on the path from the leaf node to that node. {E, I} are the descendants of the node {B}.
7. **Sibling:** Children of the same parent node are called siblings. {D, E} are called siblings.
8. **Level of a node:** The count of edges on the path from the root node to that node. The root node has level 0.
9. **Internal node:** A node with at least one child is called Internal Node.
10. **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.
11. **Subtree:** Any node of the tree along with its descendant.

```
import java.util.ArrayList;
import java.util.List;

public class TreeBasicTerminologies
       {

        # Write Code Here
       }
static void printChildren(int root, List<List<Integer>> adj)
       {
         # Write Code Here
       }
    static void printLeafNodes(int root, List<List<Integer>> adj)
       {
        # Write Code Here
       }
    static void printDegrees(int root, List<List<Integer>> adj)
       {
        # Write Code Here
       }
    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

9.2 Binary Tree Traversal Techniques

A binary tree data structure can be traversed in following ways:

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Level Order Traversal



Algorithm Inorder (tree)

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Algorithm Preorder (tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left->subtree)
3. Traverse the right subtree, i.e., call Preorder(right->subtree)

Algorithm Postorder (tree)

1. Traverse the left subtree, i.e., call Postorder(left->subtree)
2. Traverse the right subtree, i.e., call Postorder(right->subtree)
3. Visit the root.

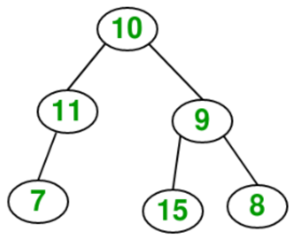
```
import java.util.Scanner;
class Node
       {
         # Write Code Here
       }
class BT {
        Node root;
        BT() {
         this.root = null;
```

```
    }
    void insert(int data)
      {
      # Write Code Here
      }
    Node insertRec(Node root, int data)
      {
        # Write Code Here
      }
    void postorder(Node root)
      {
        # Write Code Here
      }
    void preorder(Node root)
      {
        # Write Code Here
     }
    }
    void inorder(Node root)
      {
      # Write Code Here
      }
}

public class BinaryTreeTraversal {
    public static void main(String[] args)
             {
        # Write Code Here
            }
        }
    }
}
```

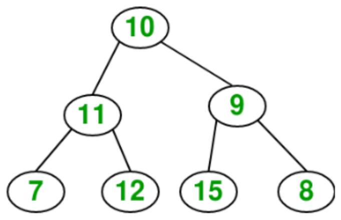
9.3 Insertion in a Binary Tree in Level Order

Given a binary tree and a key, insert the key into the binary tree at the first position available in level order.

Input: Consider the tree given below



Output:



After inserting 12

The idea is to do an iterative level order traversal of the given tree using queue. If we find a node whose left child is empty, we make a new key as the left child of the node. Else if we find a node whose right child is empty, we make the new key as the right child. We keep traversing the tree until we find a node whose either left or right child is empty.

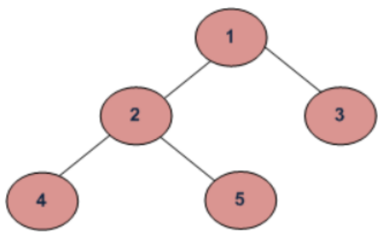
```
class Node
{
# Write Code Here
}

public class BinaryTreeInsertion
{
# Write Code Here
      }
    static Node insert(Node root, int key)
{
# Write Code Here
      }

    public static void main(String[] args)
{
        # Write Code Here
      }
}
```

9.4 Finding the Maximum Height or Depth of a Binary Tree

Given a binary tree, the task is to find the height of the tree. The height of the tree is the number of edges in the tree from the root to the deepest node.

Note: The height of an empty tree is 0.

Input: Consider the tree below



Recursively calculate the height of the left and the right subtrees of a node and assign height to the node as max of the heights of two children plus 1.

maxDepth('1') = max(maxDepth('2'), maxDepth('3')) + 1 = 2 + 1
because recursively
maxDepth('2') = max (maxDepth('4'), maxDepth('5')) + 1 = 1 + 1 and (as height of both '4' and '5' are 1)
maxDepth('3') = 1

Procedure:

- Recursively do a Depth-first search.
- If the tree is empty then return 0
- Otherwise, do the following
  - Get the max depth of the left subtree recursively i.e. call maxDepth(tree->left-subtree)
  - Get the max depth of the right subtree recursively i.e. call maxDepth(tree->right-subtree)
  - Get the max of max depths of left and right subtrees and add 1 to it for the current node.

$$max_depth = max(maxdeptofleftsubtree, maxdepthofrightsubtree) + 1$$

- Return max_depth.

```
class Node
{
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class MaximumDepthOfTree
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
```

9.5 Deletion in a Binary Tree

Given a binary tree, delete a node from it by making sure that the tree shrinks from the bottom (i.e. the deleted node is replaced by the bottom-most and rightmost node).

Input: Delete 10 in below tree

```
    10
   /  \
 20    30
```

Output:

```
    30
   /
 20
```

Input: Delete 20 in below tree

```
    10
   /  \
 20    30
         \
          40
```

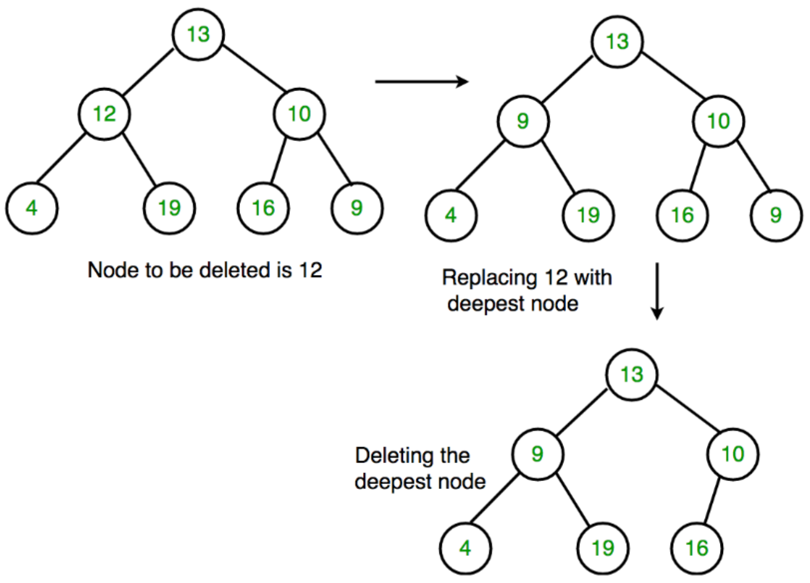
Output:

```
    10
  /    \
40      30
```

Algorithm:

1. Starting at the root, find the deepest and rightmost node in the binary tree and the node which we want to delete.
2. Replace the deepest rightmost node's data with the node to be deleted.

3. Then delete the deepest rightmost node.



```
class Node {
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTreeDeletion
{
        # Write Code Here
      }
    static void deleteDeepest(Node root, Node dNode)
{
        # Write Code Here
}
    static Node deletion(Node root, int key)
{
        # Write Code Here
}


    public static void main(String[] args)
{
        # Write Code Here
}
}
```

10. Binary Search Tree (BST)

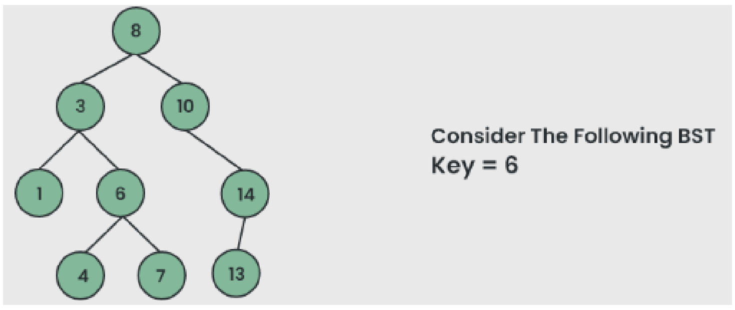
10.1 Searching in Binary Search Tree

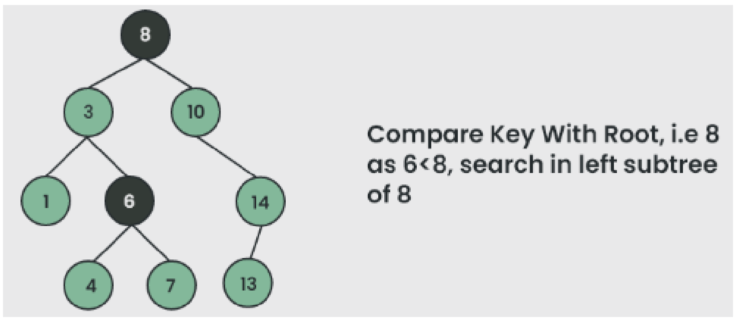
Given a BST, the task is to delete a node in this BST. For searching a value in BST, consider it as a sorted array. Perform search operation in BST using Binary Search Algorithm.

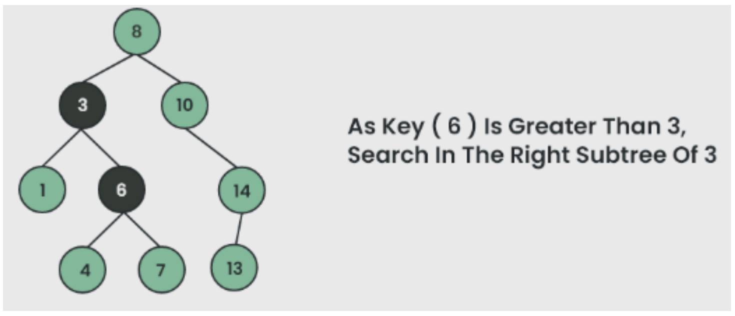
Algorithm to search for a key in a given Binary Search Tree:

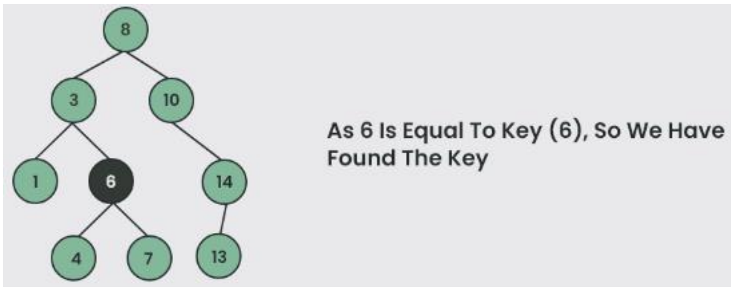
Let's say we want to search for the number **X,** We start at the root. Then:

- We compare the value to be searched with the value of the root.
    - If it's equal we are done with the search if it's smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.






8
3
10
1
6
14
4
7
13
As Key (6) Is Greater Than 3,
Search In The Right Subtree Of 3

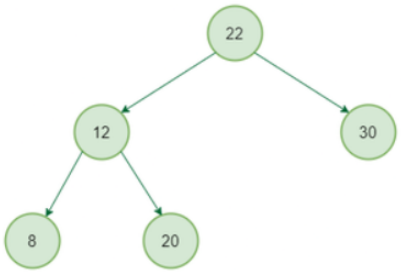


```
// Node class to represent each node of the BST
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}
class BST
{
# Write Code Here
}
    Node search(int key) {
      return searchRec(root, key);
}
    Node searchRec(Node root, int key)
      {
      # Write Code Here
      }
    public static void main(String[] args)
{
       # Write Code Here
      }
}
```

10.2 Find the node with Minimum Value in a BST

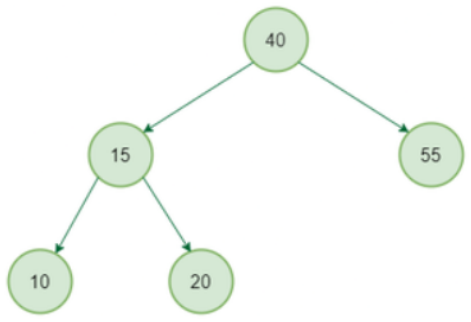
Write a function to find the node with minimum value in a Binary Search Tree.

Input: Consider the tree given below



Output: 8

Input: Consider the tree given below



Output: 10

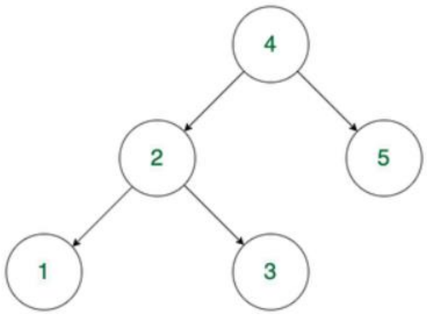
```
import java.util.ArrayList;
import java.util.List;

class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinarySearchTree
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

10.3 Check if a Binary Tree is BST or not

A binary search tree (BST) is a node-based binary tree data structure that has the following properties.

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. Both the left and right subtrees must also be binary search trees.
4. Each node (item in the tree) has a distinct key.

Input: Consider the tree given below



Output: Check if max value in left subtree is smaller than the node and min value in right subtree greater than the node, then print it "Is BST" otherwise "Not a BST"

Procedure:
1. If the current node is null then return true
2. If the value of the left child of the node is greater than or equal to the current node then return false
3. If the value of the right child of the node is less than or equal to the current node then return false
4. If the left subtree or the right subtree is not a BST then return false
5. Else return true

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree
      {
      # Write Code Here
      }

    boolean isBST(Node node) {
        return isBSTUtil(node, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    boolean isBSTUtil(Node node, int min, int max)
      {
      # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

10.4 Second Largest Element in BST

Given a Binary search tree (BST), find the second largest element.

Input: Root of below BST

```
          10
         /
        5
```

Output: 5

Input: Root of below BST

```
         10

        /  \
        5     20

                \

                 30
```

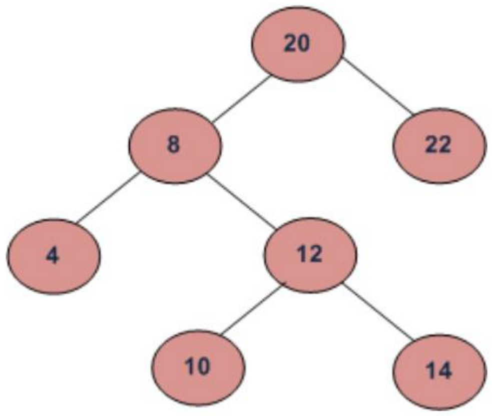
Output: 20

Procedure: The second largest element is second last element in inorder traversal and second element in reverse inorder traversal. We traverse given Binary Search Tree in reverse inorder and keep track of counts of nodes visited. Once the count becomes 2, we print the node.

```
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}

class BinarySearchTree
      {
      # Write Code Here
      }

        secondLargestUtil(node.right);
        count++;

        // If count is equal to 2 then this is the second largest
        if (count == 2) {
            System.out.println("The second largest element is " + node.key);
            return;
        }

        secondLargestUtil(node.left);
    }

    // Function to find the second largest element
    void secondLargest(Node node) {
        count = 0;
        secondLargestUtil(node);
    }

    // Driver code
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

Try:

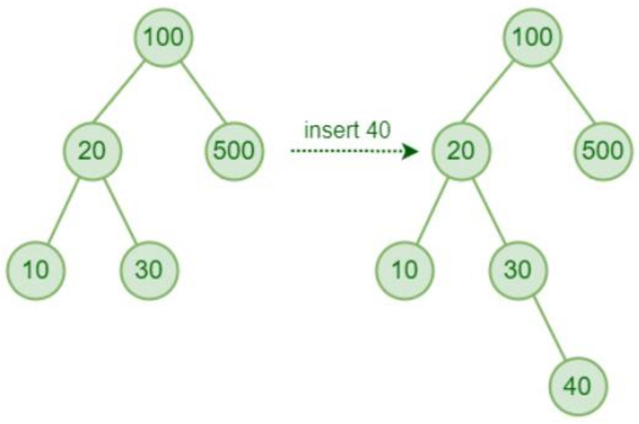
1. **Kth largest element in BST when modification to BST is not allowed:** Given a Binary Search Tree (BST) and a positive integer k, find the k'th largest element in the Binary Search Tree. For a given BST, if k = 3, then output should be 14, and if k = 5, then output should be 10.



10.5 Insertion in Binary Search Tree (BST)

Given a Binary search tree (BST), the task is to insert a new node in this BST.

Input: Consider a BST and insert the element 40 into it.



Procedure for inserting a value in a BST:

A new key is always inserted at the leaf by maintaining the property of the binary search tree. We start searching for a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. The below steps are followed while we try to insert a node into a binary search tree:

- Check the value to be inserted (say X) with the value of the current node (say val) we are in:
  - If X is less than val move to the left subtree.
  - Otherwise, move to the right subtree.
- Once the leaf node is reached, insert X to its right or left based on the relation between X and the leaf node's value.

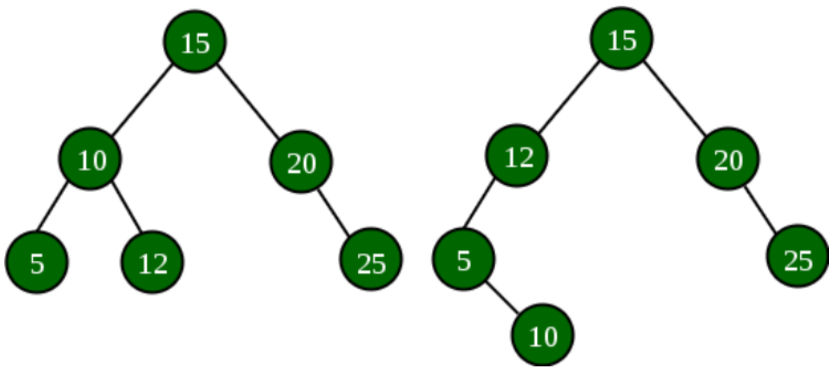
```
// A utility class that represents an individual node in a BST
class Node {
    int val;
    Node left, right;

    public Node(int item) {
        val = item;
        left = right = null;
    }
}
```

```
class BinarySearchTree
       {

       # Write Code Here

       }

    void inorder()
       {
         inorderRec(root);
       }

    void inorderRec(Node root)
       {
         # Write Code Here

         }
    }

    // Driver code
    public static void main(String[] args)
       {
         # Write Code Here
       }
}
```

Try:

1. **Check if two BSTs contain same set of elements:** Given two Binary Search Trees consisting of unique positive elements, we have to check whether the two BSTs contain the same set of elements or not.

Input: Consider two BSTs which contains same set of elements {5, 10, 12, 15, 20, 25}, but the structure of the two given BSTs can be different.



11. AVL Tree

11.1 Insertion in an AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes. To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing. Following are two basic operations that can be performed to balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

- Left Rotation
- Right Rotation

T1, T2 and T3 are subtrees of the tree, rooted with y (on the left side) or x (on the right side)

```
     y                               x
    / \     Right Rotation          /  \
   x   T3   - - - - - - - >        T1   y
  / \       < - - - - - - -            / \
T1   T2     Left Rotation            T2   T3
```

Keys in both of the above trees follow the following order
keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)
So BST property is not violated anywhere.

Procedure for inserting a node into an AVL tree

Let the newly inserted node be w

- Perform standard BST insert for w.
- Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.
- Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that need to be handled as x, y and z can be arranged in 4 ways.
- Following are the possible 4 arrangements:
  - y is the left child of z and x is the left child of y (Left Left Case)
  - y is the left child of z and x is the right child of y (Left Right Case)
  - y is the right child of z and x is the right child of y (Right Right Case)
  - y is the right child of z and x is the left child of y (Right Left Case)

```
class TreeNode {
    int val, height;
    TreeNode left, right;

    TreeNode(int d) {
        val = d;
        height = 1;
    }
}
class AVL_Tree {

    # write the code
    }

    TreeNode leftRotate(TreeNode x)
```

```
      {
      # write the code
      }
    public static void main(String[] args) {
       # write the code
      }
```

11.2 Deletion in an AVL Tree

Given an AVL tree, make sure that the given tree remains AVL after every deletion, we must augment the standard BST delete operation to perform some re-balancing. Following are two basic operations that can be performed to re-balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

1. Left Rotation
2. Right Rotation

T1, T2 and T3 are subtrees of the tree rooted with y (on left side)
or x (on right side)

```
    y                               x
   / \     Right Rotation          /  \
  x   T3   - - - - - - - >        T1   y
 / \       < - - - - - - -            / \
T1  T2     Left Rotation            T2  T3
```

Keys in both of the above trees follow the following order
   keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)
So BST property is not violated anywhere.

Procedure to delete a node from AVL tree:

Let w be the node to be deleted

1. Perform standard BST delete for w.
2. Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y. Note that the definitions of x and y are different from insertion here.
3. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
   i. y is left child of z and x is left child of y (Left Left Case)
   ii. y is left child of z and x is right child of y (Left Right Case)
   iii. y is right child of z and x is right child of y (Right Right Case)
   iv. y is right child of z and x is left child of y (Right Left Case)

```
class TreeNode
      {
      int val, height;
      TreeNode left, right;

      TreeNode(int d) {
        val = d;
        height = 1;
    }
}

class AVL_Tree {

    TreeNode leftRotate(TreeNode z)
      {
      # Write code here
      }

    TreeNode rightRotate(TreeNode z)
      {
      # Write code here
      }

      TreeNode insert(TreeNode node, int key)
      {
      # Write code here
        }
```

11.3 Count Greater Nodes in AVL Tree

Given an AVL tree, calculate number of elements which are greater than given value in AVL tree.

Input: x = 5

Root of below AVL tree

```
      9
     / \
    1   10
   / \    \
  0   5    11
 /   / \
-1  2   6
```

Output: 4

Explanation: There are 4 values which are greater than 5 in AVL tree which are 6, 9, 10 and 11.

```
class TreeNode {
    int key, height, desc;
    TreeNode left, right;

    TreeNode(int d) {
        key = d;
        height = 1;
        desc = 0;
    }
}

class AVL_Tree
      {
      # Write code here
      }

    TreeNode insert(TreeNode node, int key)
      {
      # Write code here
      }
    TreeNode minValueNode(TreeNode node)
```

```
       {
       # Write code here
       }

    TreeNode deleteNode(TreeNode root, int key)
       {
        # Write code here
       }
void preOrder(TreeNode node)
       {
       # Write code here
       }

public class Main
       {
       # Write code here
       }
}
```

11.4 Minimum Number of Nodes in an AVL Tree with given Height

Given the height of an AVL tree ‘h’, the task is to find the minimum number of nodes the tree can have.

Input: H = 0
Output: N = 1
Only '1' node is possible if the height
of the tree is '0' which is the root node.

Input: H = 3
Output: N = 7

Recursive approach:
In an AVL tree, we have to maintain the height balance property, i.e. difference in the height of the left and the right subtrees cannot be other than -1, 0 or 1 for each node.
We will try to create a recurrence relation to find minimum number of nodes for a given height, n(h).

- For height = 0, we can only have a single node in an AVL tree, i.e. n(0) = 1
- For height = 1, we can have a minimum of two nodes in an AVL tree, i.e. n(1) = 2
- Now for any height ‘h’, root will have two subtrees (left and right). Out of which one has to be of height h-1 and other of h-2. [root node excluded]
- So, n(h) = 1 + n(h-1) + n(h-2) is the required recurrence relation for h>=2 [1 is added for the root node]

```
public class AVLTreeMinimumNodes {

    public static int AVLnodes(int height)
       {
       # Write code here
      }

    public static void main(String[] args) {
        int H = 3;
        System.out.println(AVLnodes(H)); // Output: 4
    }
}
```

12. Graph Traversal

12.1 Breadth First Search

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
For a given graph G, print BFS traversal from a given source vertex.

```
import java.util.*;

public class Graph {
    private Map<Integer, List<Integer>> graph;

    public Graph()
      {
        graph = new HashMap<>();
    }

    public void addEdge(int u, int v) {
        if (!graph.containsKey(u)) {
            graph.put(u, new ArrayList<>());
        }
        graph.get(u).add(v);
    }


          public void BFS(int s)
      {
      # Write code here
      }

    public static void main(String[] args)
      {
      # Write code here
      }
}
```

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

12.2 Depth First Search

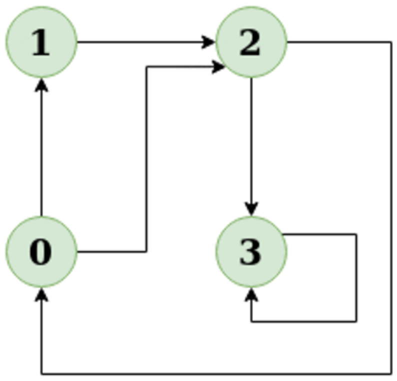
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

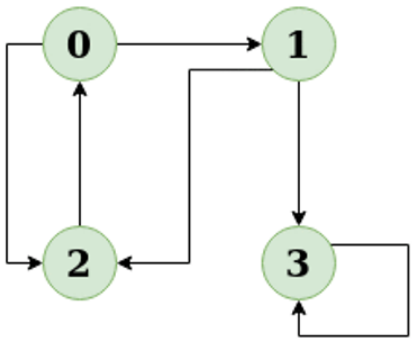
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:



```
import java.util.*;
class Graph {
    private Map<Integer, List<Integer>> graph;

    public Graph() {
        // Initialize the graph as a HashMap of ArrayLists
        graph = new HashMap<>();
    }
    public void addEdge(int u, int v)
      {
      # Write code here
      }

    public void DFS(int v) {

        DFSUtil(v, visited);
    }
    public static void main(String[] args)
      {
      # Write code here
      }
}
```

12.3 Best First Search (Informed Search)

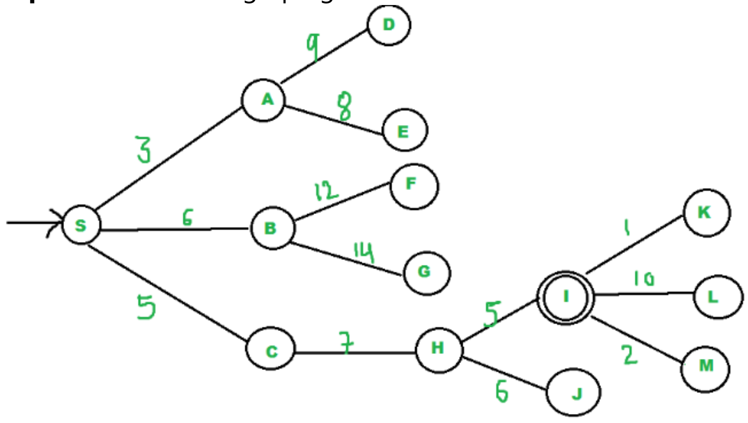
The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

Implementation of Best First Search:
We use a priority queue or heap to store the costs of nodes that have the lowest evaluation function value. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm:
Best-First-Search(Graph g, Node start)

```
  1) Create an empty PriorityQueue
     PriorityQueue pq;
  2) Insert "start" in pq.
     pq.insert(start)
  3) Until PriorityQueue is empty
       u = PriorityQueue.DeleteMin
       If u is the goal
          Exit
       Else
          Foreach neighbor v of u
             If v "Unvisited"
                 Mark v "Visited"
                 pq.insert(v)
          Mark u "Examined"
End procedure
```

Input: Consider the graph given below.



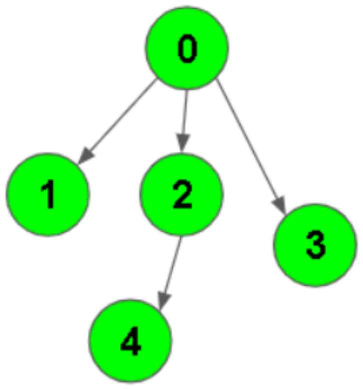
- We start from source "S" and search for goal "I" using given costs and Best First search.
- pq initially contains S
  - We remove S from pq and process unvisited neighbors of S to pq.
  - pq now contains {A, C, B} (C is put before B because C has lesser cost)
- We remove A from pq and process unvisited neighbors of A to pq.
  - pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq.
  - pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq.
  - pq now contains {H, E, D, F, G}
- We remove H from pq.
- Since our goal "I" is a neighbor of H, we return.

```
import java.util.*;

public class BestFirstSearch {
    static int v = 14;
    static List<List<Pair<Integer, Integer>>> graph = new ArrayList<>();
    static void addedge(int x, int y, int cost) {
        graph.get(x).add(new Pair<>(y, cost));
        graph.get(y).add(new Pair<>(x, cost));
    }

    static void best_first_search(int actual_Src, int target, int n)
      {
      # Write code here
      }

    public static void main(String[] args)
      {
      # Write code here
      }
}
```

12.4 Breadth First Traversal of a Graph

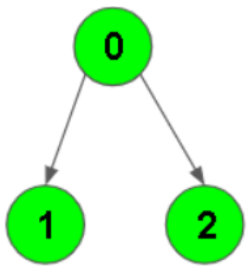
Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.
One can move from node u to node v only if there's an edge from u to v. Find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the input graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.

Input: Consider the graph given below where V = 5, E = 4, edges = {(0,1), (0,2), (0,3), (2,4)}



Output: 0 1 2 3 4
Explanation:
0 is connected to 1, 2, and 3.
2 is connected to 4.
So starting from 0, it will go to 1 then 2 then 3. After this 2 to 4, thus BFS will be 0 1 2 3 4.

Input: Consider the graph given below where V = 3, E = 2, edges = {(0, 1), (0, 2)}



Output: 0 1 2
Explanation:

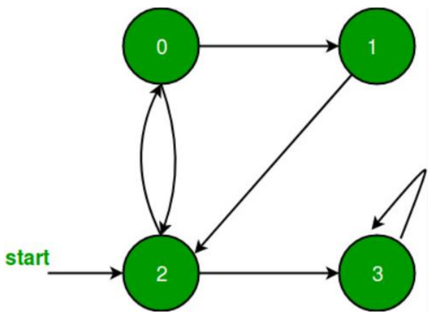
0 is connected to 1, 2. So starting from 0, it will go to 1 then 2, thus BFS will be 0 1 2.
Your task is to complete the function **bfsOfGraph()** which takes the integer V denoting the number of vertices and adjacency list as input parameters and returns a list containing the BFS traversal of the graph starting from the 0th vertex from left to right.

```
import java.util.*;

class Graph {
    private int V;
    private LinkedList<Integer>[] adj;

    Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    void addEdge(int v, int w) {
        adj[v].add(w);
    }

    void BFS(int s)
       {
       # Write Code Here
         }
 }

    public static void main(String args[]) {
       # Write Code Here
    }
}
```

12.5 Depth First Search (DFS) for Disconnected Graph

Given a Disconnected Graph, the task is to implement DFS or Depth First Search Algorithm for this Disconnected Graph.

Input: Consider the graph given below.



Output: 0 1 2 3

Procedure for DFS on Disconnected Graph:
Iterate over all the vertices of the graph and for any unvisited vertex, run a DFS from that vertex.

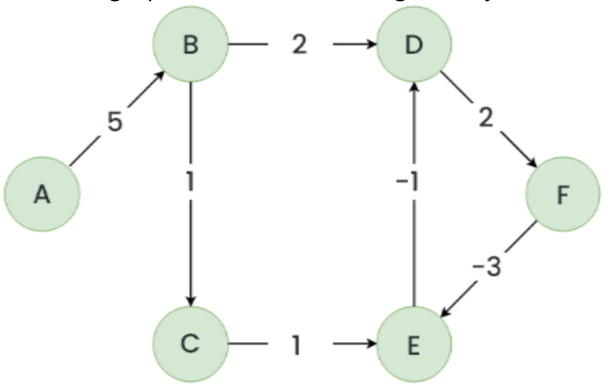
```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
// Class representing a directed graph using adjacency list representation
class Graph
       {
        # Write Code Here
       }

    public Graph()
       {
         graph = new HashMap<>();
       }

    public void addEdge(int u, int v)
       {
       # Write Code Here
       }
    private void DFSUtil(int v, boolean[] visited)
       {
       # Write Code Here
       }
    }

    public void DFS() {
        boolean[] visited = new boolean[graph.size()];
       # Write Code Here
       }

    public static void main(String[] args)
       {
       # Write Code Here
       }
}
```

Try:

1. **Detect a negative cycle in a Graph (Bellman Ford):** A Bellman-Ford algorithm is also guaranteed to find the shortest path in a graph, similar to Dijkstra's algorithm. Although Bellman-Ford is slower than Dijkstra's algorithm, it is capable of handling graphs with negative edge weights, which makes it more versatile. The shortest path cannot be found if there exists a negative cycle in the graph. If we continue to go around the negative cycle an infinite number of times, then the cost of the path will continue to decrease (even though the length of the path is increasing).

Consider a graph G and detect a negative cycle in the graph using Bellman Ford algorithm.



13. Minimum Spanning Tree (MST)

13.1 Kruskal's Algorithm

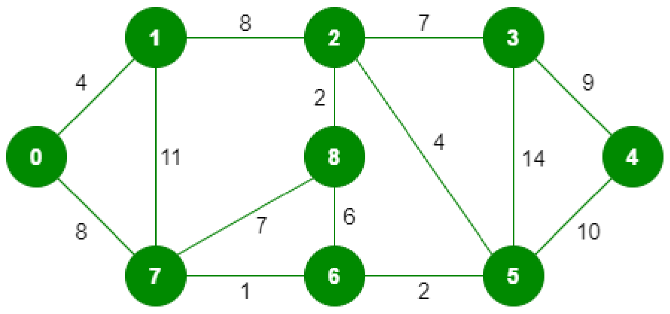
In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

MST using Kruskal's algorithm:

4. Sort all the edges in non-decreasing order of their weight.
5. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
6. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



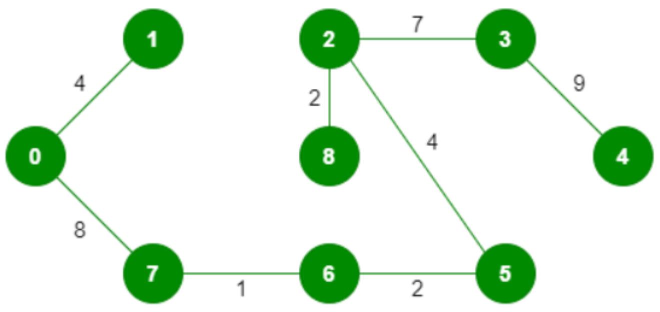
The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:



```
// Kruskal's algorithm to find minimum Spanning Tree of a given connected,
import java.util.*;

public class Graph {

    class Edge implements Comparable<Edge> {
        int src, dest, weight;

        // Comparator function used for sorting edges
        public int compareTo(Edge compareEdge) {
            return this.weight - compareEdge.weight;
        }
    }

    private int V; // Number of vertices
    private List<Edge> edges; // List of edges

    public Graph(int vertices) {
        this.V = vertices;
        this.edges = new ArrayList<>();
    }

    public void addEdge(int u, int v, int w)
      {
      # Write Code Here
      }
    private void union(int[] parent, int[] rank, int x, int y)
      {
      # Write Code Here
      }

    public void KruskalMST()
      {
      # Write Code Here
      }
 public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1, 10);
        g.addEdge(0, 2, 6);
        g.addEdge(0, 3, 5);
        g.addEdge(1, 3, 15);
        g.addEdge(2, 3, 4);

        // Function call
        g.KruskalMST();
    }
}
```

Output: Following are the edges in the constructed MST

```
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
```

13.2 Prim's Algorithm

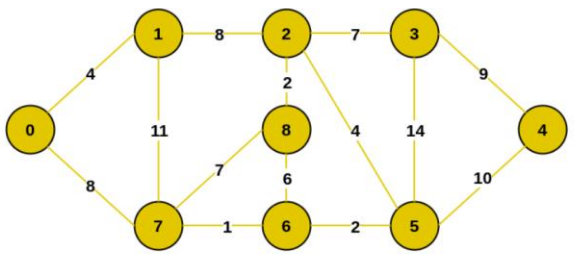
The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Prim's Algorithm:

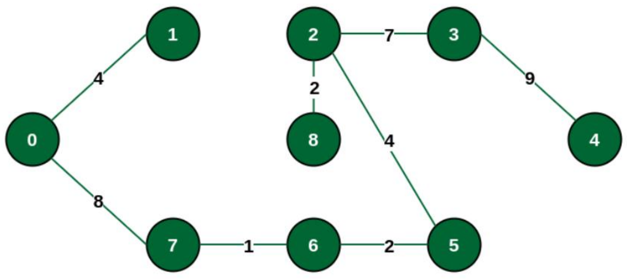
The working of Prim's algorithm can be described by using the following steps:

7. Determine an arbitrary vertex as the starting vertex of the MST.
8. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
9. Find edges connecting any tree vertex with the fringe vertices.
10. Find the minimum among these edges.
11. Add the chosen edge to the MST if it does not form any cycle.
12. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



```
import java.util.Arrays;

public class Graph {

    private int V; // Number of vertices
    private int[][] graph; // Adjacency matrix representation of graph

    public Graph(int vertices) {
        this.V = vertices;
        this.graph = new int[V][V];
    }
    public void printMST(int[] parent) {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
        }
    }
    private int minKey(int[] key, boolean[] mstSet)
      {
              # Write Code Here
      }
    public void primMST()
      {
              # Write Code Here

      }
    public static void main(String[] args)
      {
              # Write Code Here
      }
}
```

Output:

```
Edge    Weight
0 - 1     2
1 - 2     3
0 - 3     6
1 - 4     5
```

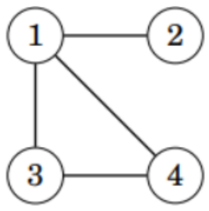
13.3 Total Number of Spanning Trees in a Graph

If a graph is a complete graph with n vertices, then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph. In complete graph, the task is equal to counting different labeled trees with n nodes for which have Cayley's formula.

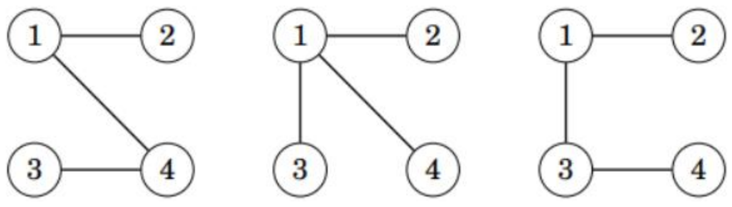
Laplacian matrix:

A Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.

Kirchhoff's theorem provides a way to calculate the number of spanning trees for a given graph as a determinant of a special matrix. Consider the following graph,



All possible spanning trees are as follows:



In order to calculate the number of spanning trees, construct a Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.
for the above graph, The Laplacian matrix will look like this

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

The number of spanning trees equals the determinant of a matrix.
The Determinant of a matrix that can be obtained when we remove any row and any column from L.
For example, if we remove the first row and column, the result will be,

$$\det(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}) = 3.$$

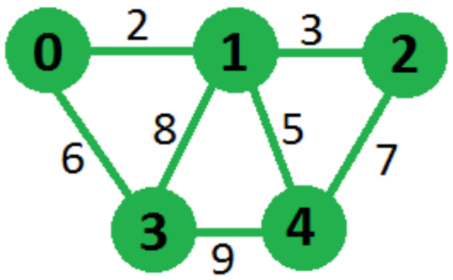
The determinant is always the same, regardless of which row and column we remove from L.

```
import java.util.Arrays;

public class NumberOfSpanningTrees {

    static final int MAX = 100;
    static final int MOD = 1000000007;
    void multiply(long[][] A, long[][] B, long[][] C, int size) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                C[i][j] = 0;
                for (int k = 0; k < size; k++) {
                    C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
                }
            }
        }
    }
 void power(long[][] A, int N, long[][] result, int size)
       {
       # Write Code Here
       }

 long numOfSpanningTree(int[][] graph, int V)
       {
       # Write Code Here
       }

    public static void main(String[] args) {
        int V = 4; // Number of vertices in graph
        int E = 5; // Number of edges in graph
        int[][] graph = { { 0, 1, 1, 1 }, { 1, 0, 1, 1 }, { 1, 1, 0, 1 }, { 1, 1, 1, 0
} };

        NumberOfSpanningTrees obj = new NumberOfSpanningTrees();
        System.out.println(obj.numOfSpanningTree(graph, V));
    }
}
```

13.4 Minimum Product Spanning Tree

A minimum product spanning tree for a weighted, connected, and undirected graph is a spanning tree with a weight product less than or equal to the weight product of every other spanning tree. The weight product of a spanning tree is the product of weights corresponding to each edge of the spanning tree. All weights of the given graph will be positive for simplicity.

Input:



Output: Minimum Product that we can obtain is 180 for above graph by choosing edges 0-1, 1-2, 0-3 and 1-4

This problem can be solved using standard minimum spanning tree algorithms like Kruskal and prim's algorithm, but we need to modify our graph to use these algorithms. Minimum spanning tree algorithms tries to minimize the total sum of weights, here we need to minimize the total product of weights. We can use the property of logarithms to overcome this problem.

log(w1* w2 * w3 * .... * wN) = log(w1) + log(w2) + log(w3) ..... + log(wN)

We can replace each weight of the graph by its log value, then we apply any minimum spanning tree algorithm which will try to minimize the sum of log(wi) which in turn minimizes the weight product.

```
import java.util.Arrays;

public class MinimumProductMST {
    // Number of vertices in the graph
    static final int V = 5;

    // A utility function to find the vertex with minimum key value, from the set of
    // vertices not yet included in MST
    int minKey(int key[], boolean mstSet[]) {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++) {
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }
        }

        return min_index;
    }

    void printMST(int parent[], int n, int graph[][])
      {
      # Write Code Here
      }
    void primMST(int inputGraph[][], int logGraph[][])
      {
      # Write Code Here
      }
    void minimumProductMST(int graph[][])
      {
      # Write Code Here
      }
    public static void main(String[] args)
      {
      # Write Code Here
    }
}
```

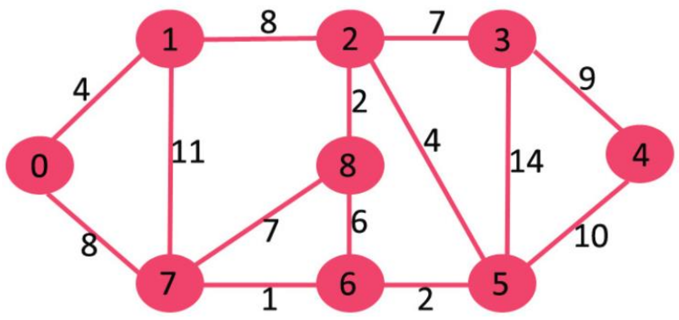
13.5 Reverse Delete Algorithm for Minimum Spanning Tree

In Reverse Delete algorithm, we sort all edges in decreasing order of their weights. After sorting, we one by one pick edges in decreasing order. We include current picked edge if excluding current edge causes disconnection in current graph. The main idea is delete edge if its deletion does not lead to disconnection of graph.

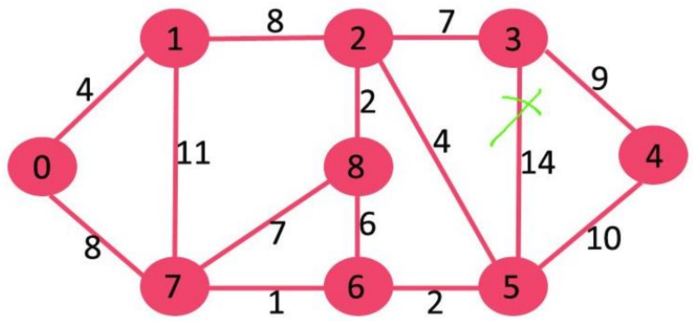
Algorithm:

1. Sort all edges of graph in non-increasing order of edge weights.
2. Initialize MST as original graph and remove extra edges using step 3.
3. Pick highest weight edge from remaining edges and check if deleting the edge disconnects the graph or not.
   If disconnects, then we don't delete the edge.
   Else we delete the edge and continue.

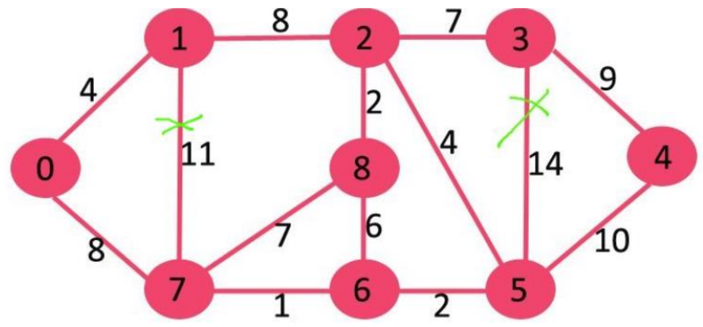
Input: Consider the graph below



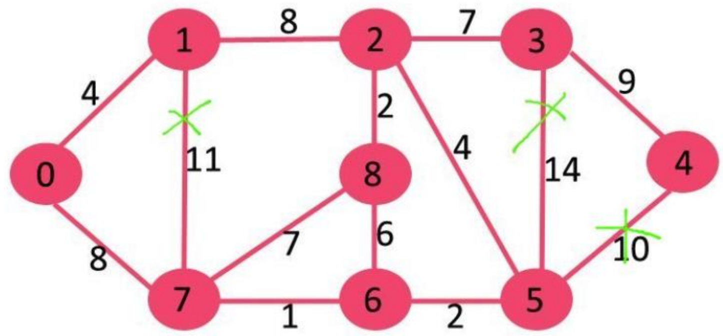
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



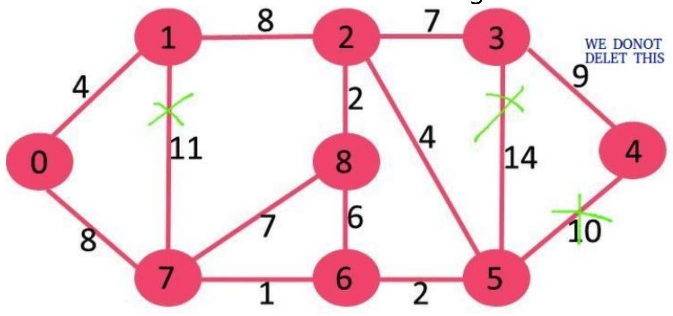
Next we delete 11 as deleting it doesn't disconnect the graph.



Next we delete 10 as deleting it doesn’t disconnect the graph.



Next is 9. We cannot delete 9 as deleting it causes disconnection.



We continue this way and following edges remain in final MST.

Edges in MST

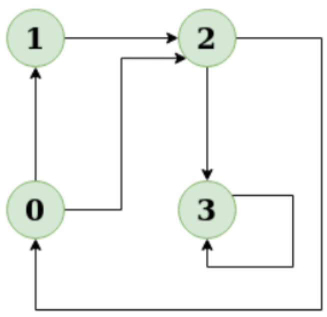
(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)

```
import java.util.ArrayList;
import java.util.Collections;

// Edge class to represent edges in the graph
class Edge {
    int src, dest, weight;

    Edge(int src, int dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }
}
class Graph
      {
      # Write Code Here
      }

    // Function to add an edge to the graph
    void addEdge(int u, int v, int w) {
      # Write Code Here
      }
    void dfs(int v, boolean[] visited)
      {
       # Write Code Here
      }

    // Function to check if the graph is connected
    boolean connected()
      {
      # Write Code Here
      }
void reverseDeleteMST()
      {
       # Write Code Here
      }
}

public class ReverseDeleteMST {
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

Try:

1. **Detect Cycle in a Directed Graph:** Given the root of a Directed graph, The task is to check whether the graph contains a cycle or not.

Input: N = 4, E = 6



Output: Yes

Explanation: The diagram clearly shows a cycle 0 -> 2 -> 0

14. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	**Minimum number of problems to solve**	**Required score**
• CodeChef	20	200
• Leetcode	20	200
• GeeksforGeeks	20	200
• SPOJ	5	50
• InterviewBit	10	1000
• Hackerrank	25	250
• Codeforces	10	100
• BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certification:

2. HackerRank - Problem Solving Skills Certification (Basic and Intermediate)
2. GeeksforGeeks – Data Structures and Algorithms Certification
3. CodeChef - Learn Data Structures and Algorithms Certification
4. Interviewbit – DSA pro / Python pro
5. Edx – Data Structures and Algorithms
5. NPTEL – Programming, Data Structures and Algorithms
6. NPTEL – Introduction to Data Structures and Algorithms
7. NPTEL – Data Structures and Algorithms
8. NPTEL – Programming and Data Structure

V. TEXT BOOKS:

1. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.

VI. REFERENCE BOOKS:

1. S. Lipschutz, “Data Structures”, Tata McGraw Hill Education, 1st Edition, 2008.
2. D. Samanta, “Classic Data Structures”, PHI Learning, 2nd Edition, 2004.

VII. ELECTRONICS RESOURCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

VIII. MATERIALS ONLINE

1. Course Content
2. Laboratory manual

COURSE CONTENT

PROGRAMMING WITH OBJECTS LABORATORY								
III Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
AITD02	**Core**	-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: **Essentials of Problem Solving**								

I. COURSE OVERVIEW:

Windows, a dominant OS, provides a user-friendly platform and robust development environ- ment via the .NET Framework. Developers leverage C#, a versatile language on .NET, for diverse applications across desktops, servers, web, mobile, and IoT. C# accelerates development across the software lifecycle, from enterprise solutions to responsive web apps. This lab course equips students with core C# skills and practical experience in real-world scenarios, preparing them for success in software development.

II. COURSE OBJECTIVES:

The students will try to learn

I. The basic concepts of object oriented programming.
II. The application of object oriented features for developing flexible and extensible applications.
III. The Graphical User Interface (GUI) with database connectivity to develop web applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 **Core Programming Skills:** Develop core programming skills in C# for basic applica- tion development.
CO 2 **Function Design and Implementation:** Design and implement functions to enhance code modularity and reusability.
CO 3 **Advanced Text and Service Integration:** Utilize advanced techniques for pattern matching and integrating external services.
CO 4 **Object-Oriented Design:** Apply principles of object-oriented design to create struc- tured and scalable applications.
CO 5 **Error Handling and File Management:** Implement robust error handling and manage file operations effectively.
CO 6 **Concurrency Management:** Create and manage concurrent processes to improve ap- plication performance.

IV. COURSE CONTENT:

Introduction to the Course C# & .NET

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

Course Overview

Windows, a dominant OS, provides a user-friendly platform and robust development environment via the .NET Framework. Developers leverage C#, a versatile language on .NET, for diverse applications across desktops, servers, web, mobile, and IoT. C# accelerates development across the software lifecycle, from enterprise solutions to responsive web apps. This lab course equips students with core C# skills and practical experience in real-world scenarios, preparing them for success in software development.

Software Requirements

Windows 10 Pro or Enterprise is recommended for optimal performance.
While Windows 10 Home is sufficient for introductory C# courses, advanced courses re- quire Windows 10 Pro or higher.

Visual Studio Community Edition 2017 or later is required and available for free download from Microsoft's official website: `https://www.visualstudio.com/`

Hardware Requirements

- Intel/AMD multi-core processor (i3 or better)
- Minimum 8GB RAM, 16GB preferred
- At least 50GB of free hard drive space for Visual Studio and project files

Prerequisites

Completion of "Visual Studio" or equivalent experience is recommended. No prior programming knowledge or experience is necessary.

Next Steps

Upon completion of this course, consider taking C# Programming certification

IV. COURSE CONTENT

Getting Started Exercises

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1.1 Installing Visual Studio and Setting up the .NET Environ- ment

The objective of this lab is to guide students through the installation of Visual Studio and the setup of the .NET environment necessary for C# programming.

Requirements

- A computer with internet access.
- Windows operating system (recommended) or macOS.
- Administrative privileges to install software.

Steps to Install Visual Studio

1. **Download Visual Studio:**
   - Visit the official Visual Studio website: `https://visualstudio.microsoft. com/`
   - Click on the “Download” button for the Community edition (free version).
2. **Run the Installer:**
   - Once the download is complete, run the installer.
   - Follow the on-screen instructions to continue with the installation.
3. **Select Workloads:**
   - In the installer, you will see a list of workloads. Select *“.NET desktop development”*.
   - Optionally, you can select other workloads if needed for future use, such as *“ASP.NET and web development”*.
4. **Install:**
   - Click on the “Install” button to begin the installation process.
   - Wait for the installation to complete. This may take some time depending on your internet speed and system performance.
5. **Launch Visual Studio:**
   - Once the installation is complete, launch Visual Studio from the Start menu or desk- top shortcut.
   - Sign in with your Microsoft account if prompted.
   - Choose the default environment settings when prompted.

Setting up the .NET Environment

1. **Create a New Project:**
   - Click on "Create a new project" from the Visual Studio start page.
   - Select the template *"Console App (.NET Core)"*.
   - Click "Next".
2. **Configure Your Project:**
   - Enter a name for your project (e.g., *"HelloWorld"*).
   - Choose a location to save your project.
   - Click "Create".
3. **Verify the Setup:**
   - Once the project is created, you will see the *"Program.cs"* file open in the code editor.
   - Replace the existing code with the following:

```
using System;

namespace HelloWorld
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine("Hello, World!");
}
}
}
```

   - Press `Ctrl + F5` to run the application.
   - Verify that the output window displays *"Hello, World!"*.

By completing this lab, students will have successfully installed Visual Studio, set up the .NET environment, and created their first C# console application. This foundational setup is crucial for all future labs and projects in this course.

1.2 Distance based on Speed and Time

Calculate the distance traveled by an object based on the speed (v) and time (t) entered by the user. The formula to calculate distance (d) is given by:

$$d = v \times t$$

Input

Your program should prompt the user to enter the following:

- The speed of the object in meters per second (m/s).
- The time duration in seconds (s) for which the object traveled at that speed. The program should then compute and display the distance traveled by the object.

Requirements

- Use appropriate data types for variables.
- Ensure the program handles decimal values for speed and time.
- Include error handling for non-positive values of speed or time (optional).

Sample Output

Example 1:

```
Enter the speed of the object
(m/s): 10.5 Enter the time
duration (s): 8.2

Distance traveled: 86.1 meters
```

Example 2:

```
Enter the speed of the object
(m/s): 0 Enter the time
duration (s): 5.5

Invalid input: Speed must be greater than zero.
```

1.3 Cube Root

Calculate the cube root of a given number.

Input

The program should:

- Prompt the user to enter a number.
- Calculate the cube root of the number.
- Display the result.

Requirements

- Implement a function `CalculateCubeRoot(double number)` that returns the cube root of the given number.
- Ensure the program handles valid numerical inputs.

Sample Output

Example 1:

```
Enter the
number: 27
Cube root of
27 is 3
```

Example 2:

```
Enter the
number: 64
Cube root of
64 is 4
```

Example 3:

```
Enter the
number: -8
Cube root of
-8 is -2
```

1.4 Random Number Genrator

Generate and display a sequence of random numbers.

Input

- Use the `System.Random` class for generating random numbers.
- Prompt the user to enter the number of random numbers to generate.
- Specify a range for the random numbers (e.g., between 1 and 100).
- Display each generated random number in the sequence.

Your program should ensure that each random number generated is unique within the spec- ified range.

Requirements

- Use appropriate data types and structures to store and display the random numbers.
- Handle user input validation to ensure the number of random numbers is positive.
- Implement error handling for any unexpected inputs or conditions (optional).

Sample Output

Example:

```
Enter the number of random numbers to
generate: 5 Enter the minimum value of
the range: 1
Enter the maximum value of the range: 100

Generated random numbers:
45 12 89 27 56
```

1.5 Nullable Data Types

Demonstrates the use of Nullable data types. Nullable data types allow variables to have an additional value, `null`, which represents undefined or unknown values.

Input

- Declare and initialize nullable variables of different data types (`int?`, `float?`, `bool?`).
- Assign both `null` values and non-null values to these nullable variables.
- Display the values of these nullable variables using formatted output.
- Illustrate how nullable types handle initialization and assignment.

Ensure your program includes comments or documentation to explain the purpose of each variable and the behavior of nullable data types.

Requirements

- Use appropriate syntax for declaring and initializing nullable data types in C#.
- Demonstrate the advantages of using nullable data types, such as handling database null values or optional parameters.
- Include formatted output to clearly display the values of nullable variables.

Sample Output

```
Nullable
Integers   :   ,
786    Nullable
Floats  : 3.14,
Nullable
boolean  :
```

1.6 Permutations (nPr)

Calculate the number of permutations (nPr) of a given set of `n` items taken `r` at a time. The formula to calculate permutations is given by:

$$nPr = \frac{n!}{(n-r)!}$$

where `n!` denotes the factorial of `n`.

Input

- Prompt the user to enter two integers `n` and `r`.
- Calculate the number of permutations using the provided formula.
- Display the result.

Requirements

- Implement a function `Factorial(int x)` that returns the factorial of `x`.
- Implement a function `CalculatePermutations(int n, int r)` that returns the number of permutations.
- Ensure the program handles edge cases such as `r > n` by displaying an appropriate message.

Sample Output

Example 1:

```
Enter the value of n: 5
Enter the value of r: 3
Number of permutations (5P3) = 60
```

Example 2:

```
Enter the value of n: 6
Enter the value of r: 6
Number of permutations (6P6) = 720
```

Example 3:

```
Enter the value of n: 4
Enter the value of r: 5
Error: r cannot be greater than n.
```

1.7 Binary Sum

Write a C# program to add two binary numbers.

Input

- Prompt the user to enter two binary numbers as strings.
- Validate the input to ensure that it contains only binary digits (0 and 1).
- Perform the binary addition.
- Display the sum as a binary number.

Requirements

- Implement a function `IsBinary(string binary)` that returns `true` if the string contains only binary digits, otherwise returns `false`.
- Implement a function `AddBinary(string a, string b)` that performs the binary addition and returns the result as a string.
- Ensure the program handles cases where the input is not valid binary numbers by displaying an appropriate message.

Sample Output

Example 1:

```
Enter the first binary number: 101
Enter the second binary number: 110 Sum
of 101 and 110 is 1011
```

Example 2:

```
Enter the first binary number: 1001
Enter the second binary number: 11 Sum
of 1001 and 11 is 1100
```

Example 3:

```
Enter the first binary number: 1010
Enter the second binary number: 102
Error: One or both inputs are not valid binary numbers.
```

1.8 Explore Bitwise Operators

Demonstrate the use of bitwise operators.

Input

The program should:

- Prompt the user to enter two integers.
- Perform and display the result of the following bitwise operations on the entered integers:
  - AND (&)
  - OR (|)
  - XOR (^)
  - NOT (~)
  - Left Shift (<<)
  - Right Shift (>>)

Requirements

- Implement functions for each of the bitwise operations.
- Ensure the program handles valid integer inputs.

Sample Output

Example:

```
Enter the first integer: 12
Enter the second integer: 5

Bitwise AND (12 & 5) = 4
Bitwise OR (12 | 5) = 13
Bitwise XOR (12 ^ 5) = 9
Bitwise NOT (~12) = -13 Left
Shift (12 << 2) = 48
Right Shift (12 >> 2) = 3
```

1.9 No Math

The program should

- Calculate the square root of a given number without using the built-in `Math.Sqrt()` method.

- Find the absolute value of a given number without using the built-in `Math.Abs()` method.

Requirements

- Implement a function `CalculateSquareRoot(double number)` that returns the square root of the given number using an iterative approach (e.g., Newton's method).

- Implement a function `FindAbsoluteValue(double number)` that returns the absolute value of the given number.

- Ensure the program handles valid numerical inputs.

Sample Output

Example 1:

```
Enter a number to find its square root: 16
Square root of 16 is approximately: 4

Enter a number to find its absolute value: -8.5
Absolute value of -8.5 is: 8.5
```

Example 2:

```
Enter a number to find its square root: 20
Square root of 20 is approximately: 4.4721

Enter a number to find its absolute value: 3.7
Absolute value of 3.7 is: 3.7
```

1.10 Edge cases

Explore edge cases using the `Math.Pow()` method. The program should:

- Calculate and display the results for the following scenarios:
  1. Calculate `Math.Pow(double.PositiveInfinity, 2)`.
  2. Calculate `Math.Pow(double.NegativeInfinity, 2)`.
  3. Calculate `Math.Pow(double.MinValue, 0)`.
  4. Calculate `Math.Pow(double.NaN, 2)`.
- Print the results of each calculation.

Requirements

- Implement a C# program that performs the above calculations using the `Math.Pow()` method.

- Ensure the program handles special cases such as infinity and NaN appropriately.

Sample Output

Example Output:

```
Number1 : Infinity
Number2 : Infinity
Number3 : 1 Number4
: NaN
```

Try

1. Calculate `Math.Pow(0, 0)`.
2. Calculate `Math.Pow(10, -1)`.
3. Calculate `Math.Pow(1, double.MaxValue)`.

Arrays

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

2.1 Arrow Pattern

Create a function `Arrow(num)` that generates a pattern as a 2D array for a given number `num`.

- For `Arrow(3)`, the function should return `[">", ">>", ">>>", ">>", ">"]`.
- For `Arrow(4)`, the function should return `[">", ">>", ">>>", ">>>>", ">>>>", ">>>", ">>", ">"]`.

Criteria

- The function argument `num` will always be a number greater than 0.
- Odd numbers will produce a pattern with a single “peak”.
- Even numbers will produce a pattern with two “peaks”.

Hints

- Identify that the pattern consists of increasing and then decreasing sequences of arrows.
- Create a loop to generate the increasing part of the pattern from 1 up to `num`.
- Create a second loop to generate the decreasing part of the pattern from `num-1` down to 1.
- Use a list to collect the pattern strings and then convert it to an array before returning.
- Consider how the pattern behaves for the smallest values of `num` (like 1).

2.2 Working 9 to 5

Write a function that calculates overtime and pay associated with overtime.

Working Hours

- Working hours: 9 AM to 5 PM (regular hours).
- After 5 PM is considered overtime.

Input

Your function receives an array with 4 values:

- Start of working day, in decimal format (24-hour day notation).
- End of working day (same format).
- Hourly rate.
- Overtime multiplier.

Output

Your function should return the total earnings for that day, rounded to the nearest hundredth, formatted as:

$ *earned that day*

Testcases

- `OverTime([9, 17, 30, 1.5])` ⇒ `"$240.00"`
- `OverTime([16, 18, 30, 1.8])` ⇒ `"$84.00"`
- `OverTime([13.25, 15, 30, 1.5])` ⇒ `"$52.50"`

Explanation of the Second Example

- From 16:00 to 17:00 is regular hours: $1 \times 30 = 30$

- From 17:00 to 18:00 is overtime: $1 \times 30 \times 1.8 = 54$

- Total earnings: $30 + 54 = 84 \Rightarrow$ \$84.00

Hints

- Identify the regular working hours (9 AM to 5 PM) and determine if the work falls within these hours or extends into overtime.
- Calculate the total number of regular hours and overtime hours based on the start and end times.
- Compute the earnings for regular hours and overtime hours separately.
- Use the hourly rate for regular hours and apply the overtime multiplier for overtime hours to calculate the total earnings.
- Round the final earnings to the nearest hundredth and format the result as a monetary value.

2.3 New Driving License

You have to get a new driver’s license. You show up at the office at the same time as four other people. The office says they will see everyone in alphabetical order and it takes 20 minutes for them to process each new license.

All of the agents are available now, and they can each see one customer at a time. How long will it take for you to walk out with your new license?

Your input will be a string of your name me, an integer of the number of available agents, and a string of the other four names separated by spaces others.

Return the number of minutes it will take to get your license.

Examples

```
License("Eric", 2, "Adam Caroline Rebecca
                   Frank") ⇒ 40
```

As you are in the second group of people.

```
License("Zebediah", 1, "Bob Jim Becky Pat")
                    ⇒ 100
```

As you are the last person.

```
License("Aaron", 3, "Jane Max Olivia Sam") ⇒ 20
```

As you are the first.

Hints

- Combine your name with the list of other names and sort them alphabetically to determine the order of processing.
- Determine your position in the sorted list to find out when you will be processed.
- Calculate the number of people before you in the sorted list to determine how many people will be processed before you.
- Given the number of available agents, distribute the workload so that each agent handles one person at a time.
- Compute the time it will take for you to be processed based on the number of people ahead of you and the processing time per person.

2.4 Who Won the League?

The 2019/20 season of the English Premier League (EPL) saw Liverpool FC win the title for the first time despite their bitter rivals, Manchester United, having won 13 titles!

Create a function that receives an alphabetically sorted array of the results achieved by each team in that season and the name of one of the teams. The function should return a string giving the final position of the team, the number of points achieved, and the goal difference (see examples for precise format).

The results table is given in the following format:

Team	Played	Won	Drawn	Lost	G/Diff
Arsenal	38	14	14	10	8
Aston Villa	38	9	8	21	-26
Bournemouth	38	9	7	22	-26
Brighton	38	9	14	15	-15
Burnley	38	15	9	14	-7
Chelsea	38	20	6	12	15
Crystal Palace	38	11	10	17	-19
Everton	38	13	10	15	-12
Leicester City	38	18	8	12	26
Liverpool	38	32	3	3	52
Manchester City	38	26	3	9	67
Manchester Utd	38	18	12	8	30
Newcastle	38	11	11	16	-20
Norwich	38	5	6	27	-49
Sheffield Utd	38	14	12	12	0
Southampton	38	15	7	16	-9
Tottenham	38	16	11	11	14
Watford	38	8	10	20	-28
West Ham	38	10	9	19	-13
Wolves	38	15	14	9	11

Examples

```
string[] table = [
"Arsenal, 38, 14, 14, 10, 8",
"Aston Villa, 38, 9, 8, 21, -26",
...
"West Ham, 38, 10, 9, 19, -13",
"Wolves, 38, 15, 14, 9, 11"
]

EPLResult(table, "Liverpool") -->
"Liverpool came 1st with 99 points and a goal difference of 52!"

EPLResult(table, "Manchester Utd")-->
"Manchester Utd came 3rd with 66 points and a goal difference of 30!"

EPLResult(table, "Norwich") -->
"Norwich came 20th with 21 points and a goal difference of -49!"
```

Hints

- Each result in the source table is a comma separated string.
- Score 3 points for a win and 1 point for a draw.

- If teams are tied on points, their position is determined by better goal difference.
- The input table should be considered immutable - do not make any changes to it!

2.5 The Day Rob Was Born in Dutch

I was born on the 21st of September in the year of 1970. That was a Monday. Where I was born that weekday is called *maandag*.

Write a method that, when passed a date as `year/month/day`, returns the date's weekday name in the Dutch culture. The culture identifier to use is `"nl-NL"`, not `"nl-BE"`.

You can assume the specified date is valid. Looking at the tests and doing a switch statement or similar is considered cheating. `System.Globalization.CultureInfo` should be used.

The method may be used to get the name of the Dutch weekday of other memorable days too, like in the examples below:

Examples

```
WeekdayRobWasBornInDutch(1970, 9, 21) --> "maandag"

WeekdayRobWasBornInDutch(1945, 9, 2) --> "zondag"

WeekdayRobWasBornInDutch(2001, 9, 11) --> "dinsdag"
```

Hints

- Use the `System.Globalization.CultureInfo` class to handle Dutch culture settings.
- Create a `DateTime` object using the provided year, month, and day.
- Use the `ToString` method of the `DateTime` object with the Dutch culture to get the weekday name in Dutch.
- Ensure that you specify the culture identifier `"nl-NL"` to get the correct Dutch weekday names.

2.6 Smallest Missing Positive Integer

Given an array of integers, return the smallest positive integer not present in the array. Here is a representative example. Consider the array:

```
{ -2, 6, 4, 5, 7, -1, 7, 1, 3, 6, 6, -2, 9, 10, 2, 2 }
```

After reordering, the array becomes:

```
{ -2, -2, -1, 1, 2, 2, 3, 4, 5, 6, 6, 6, 7, 7, 9, 10 }
```

... from which we see that the smallest missing positive integer is 8.

Examples

```
MinMissPos({ -2, 6, 4, 5, 7, -1, 1, 3, 6, -2, 9, 10, 2, 2 })
-> 8
// After sorting, the array becomes
// { -2, -2, -1, 1, 2, 2, 3, 4, 5, 6, 6, 7, 9, 10 }
// So the smallest missing positive integer is 8

MinMissPos({ 5, 9, -2, 0, 1, 3, 9, 3, 8, 9 })
-> 2
// After sorting, the array becomes
// { -2, 0, 1, 3, 3, 5, 8, 9, 9, 9 }
// So the smallest missing positive integer is 2

MinMissPos({ 0, 4, 4, -1, 9, 4, 5, 2, 10, 7, 6, 3, 10, 9 })
-> 1
// After sorting, the array becomes
// { -1, 0, 2, 3, 4, 4, 4, 5, 6, 7, 9, 9, 10, 10 }
// So the smallest missing positive integer is 1
```

Hints

For the sake of clarity, recall that 0 is not considered to be a positive number.

2.7 Briefcase Lock

A briefcase has a 4-digit rolling-lock. Each digit is a number from 0-9 that can be rolled either forwards or backwards.

Create a function that returns the smallest number of turns it takes to transform the lock from the current combination to the target combination. One turn is equivalent to rolling a number forwards or backwards by one.

To illustrate:

```
current-lock: 4089
target-lock: 5672
```

What is the minimum number of turns it takes to transform 4089 to 5672?

```
  4 -> 5
  4 -> 5 // Forward Turns: 1 <- Min
4 -> 3 -> 2 -> 1 -> 0 -> 9 -> 8 -> 7 -> 6 -> 5  // Backward Turns: 9

0 -> 6
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6  // Forward Turns: 6
4  <- Min
0 -> 9 -> 8 -> 7 -> 6            // Backward Turns:

8 -> 7
-> 7  // Forward Turns: 9
8 -> 9 -> 0 -> 1 -> 2         -> 3 -> 4 -> 5 -> 6
8 -> 7  // Backward Turns: 1  <- Min

9 -> 2
```

```
9 -> 0 -> 1 -> 2  // Forward Turns: 3  <- Min
9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2  // Backward Turns: 7

It takes 1 + 4 + 1 + 3 = 9 minimum turns to change the lock from 4089
to 5672.
```

Examples

```
MinTurns("4089", "5672") -> 9

MinTurns("1111", "1100") -> 2

MinTurns("2391", "4984") -> 10
```

Hints

- Both locks are in string format.
- A 9 rolls forward to 0, and a 0 rolls backwards to a 9.

2.8 Keeping Count

Given a number, create a function which returns a new number based on the following rules:

- For each digit, replace it by the number of times it appears in the number.
- The final instance of the number will be an

  integer, not a string. The following is a working

  example:

```
DigitCount(136116) -> 312332
```

Explanation:

- The number 1 appears thrice, so replace all 1s with 3s.
- The number 3 appears only once, so replace all 3s with 1s.
- The number 6 appears twice, so replace all 6s with 2s.

Examples

- DigitCount(221333) -> 221333
- DigitCount(136116) -> 312332
  - DigitCount(2) -> 1

Hints

- Each test will have a positive whole number in its parameter.

2.9 Break Point

A number has a breakpoint if it can be split in a way where the digits on the left side and the digits on the right side sum to the same number.
For instance, the number 35190 can be sliced between the digits 351 and 90, since 3 + 5 + 1 = 9 and 9 + 0 = 9. On the other hand, the number 555 does not have a breakpoint (you must split between digits).
Create a function that returns true if a number has a breakpoint, and false otherwise.

Examples

- BreakPoint(159780) → true
- BreakPoint(112) → true
- BreakPoint(1034) → true
- BreakPoint(10) → false
- BreakPoint(343) → false

Hints

- Read each digit as only one number.
- Check the Resources tab for a hint.

2.10 ATM Transactions

Implement an ATM (Automated Teller Machine) transaction program in C# that simulates basic banking operations. The program should:

- Begin by prompting the user to enter a PIN to access the ATM services.
- Validate the entered PIN against a predefined PIN (e.g., 4040).
- If the PIN is correct, provide the following options to the user:
  1. Check balance: Display the current balance in the account.
  2. Withdraw money: Prompt the user to enter the amount to withdraw. Ensure withdrawal amount is in multiples of 100 and does not exceed the available balance.
  3. Deposit money: Prompt the user to enter the amount to deposit and update the current balance accordingly.
  4. Change PIN: Allow the user to change their PIN by verifying the current PIN and entering a new PIN.
- Handle incorrect PIN entries and invalid user inputs appropriately.

Requirements

- Implement the ATM program in C# with proper validation and error handling.
- Ensure that withdrawal amounts are multiples of 100 and do not exceed the available balance.

- Display appropriate messages for each transaction and user interaction.

Sample Output

```
Enter the pin:
> 4040
  1. To check balance
  2. To withdraw money
  3. To deposit money
  4. To
  change
  the pin
  Enter
  your
  choice:
> 1
The current balance in your account is 10000

Enter the pin:
> 4040
  1. To check balance
  2. To withdraw money
  3. To deposit money
  4. To
  change
  the pin
  Enter
  your
  choice:
> 2
Enter the amount to withdraw:
> 5000
Please collect the
cash: 5000 The
current balance is
now: 5000

Enter the pin:
> 4040
  1. To check balance
  2. To withdraw money
  3. To deposit money
  4. To
  change
  the pin
  Enter
  your
  choice:
> 3
Enter the amount to be deposited:
> 2000
The current balance in the account is 7000
```

```
Enter the pin:
> 4040
  1. To check balance
  2. To withdraw money
  3. To deposit money
  4. To
  change
  the pin
  Enter
  your
  choice:
> 4
Want to change
your pin Enter
your previous
pin:
> 4040
Enter your new pin:
> 1234
Your pin is changedEnter the pin:
> 1234
  1. To check balance
  2. To withdraw money
  3. To deposit money
  4. To
  change
  the pin
  Enter
  your
  choice:
> 5
Please select correct option
```

Hints

- Test the program with incorrect PIN entries.
- Test withdrawing amounts that are not multiples of 100.
- Test depositing different amounts and verifying the updated balance.

Strings Manipulations

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

3.1 Find the Bomb

Create a function that finds the word *"bomb"* in the given string (not case sensitive). If found, return **"Duck!!!"**, otherwise, return **"There is no bomb, relax."**.

Examples

```
Bomb("There is a bomb.")
-> "Duck!!!"

Bomb("Hey, did you think there is a bomb?")
-> "Duck!!!"

Bomb("This goes boom!!!")
-> "There is no bomb, relax."
```

Hints

- "bomb" may appear in different cases (i.e., uppercase, lowercase, mixed).

3.2 Finding Nemo

Finding Nemo

You're given a string of words. You need to find the word *"Nemo"*, and return a string like this: **"I found Nemo at [the order of the word you**

find Nemo]!". If you can't find Nemo, return **"I**

can't find Nemo :(".

Examples

```
FindNemo("I am finding Nemo !")
-> "I found Nemo at 4!"

FindNemo("Nemo is me")
-> "I found Nemo at 1!"
```

```
FindNemo("I Nemo am")
-> "I found Nemo at 2!"
```

Hints

- ! , ? . are always separated from the last word.
- Nemo will always look like Nemo, and not NeMo or other capital variations.
- Nemo’s, or anything that says Nemo with something behind it, doesn’t count as Finding Nemo.
- If there are multiple Nemo’s in the sentence, only return the first one.

3.3 A Week Later

Create a function which takes in a date as a string, and returns the date a week after.

Examples

```
WeekAfter("12/03/2020")
-> "19/03/2020"

WeekAfter("21/12/1989")
-> "28/12/1989"

WeekAfter("01/01/2000")
-> "08/01/2000"
```

Hints

Note that dates will be given in **day/month/year** format. Single digit numbers should be zero padded.

3.4 Track the Robot

A robot has been given a list of movement instructions. Each instruction is either *left*, *right*, *up* or *down*, followed by a distance to move. The robot starts at [0, 0]. You want to calculate where the robot will end up and return its final position as an array.
To illustrate, if the robot is given the following instructions:

```
new string[] { "right 10", "up 50", "left 30", "down 10" }
```

It will end up 20 left and 40 up from where it started, so we return `int[] { -20, 40 }`.

Examples

```
TrackRobot(new string[] { "right 10", "up 50", "left 30", "down 10" })
-> int[] { -20, 40 }

TrackRobot(new string[] { })
-> int[] { 0, 0 }
// If there are no instructions, the robot doesn’t move.
```

```
TrackRobot(new string[] { "right 100", "right 100", "up 500", "up 10000" })
-> int[] { 200, 10500 }
```

Hints

- The only instructions given will be *left*, *right*, *up* or *down*.
- The distance after the instruction is always a positive whole number.

3.5 True Alphabetical Order

Create a function which takes every letter in every word, and puts it in alphabetical order. Note how the original word lengths must stay the same.

Examples

```
TrueAlphabetic("hello world")
-> "dehll loorw"

TrueAlphabetic("edabit is awesome")
-> "aabdee ei imosstw"

TrueAlphabetic("have a nice day")
-> "aaac d eehi nvy"
```

Hints

- All sentences will be in lowercase.
- No punctuation or numbers will be included in the Tests.

3.6 Longest Common Ending

Write a function that returns the longest common ending between two strings.

Examples

```
LongestCommonEnding("multiplication", "ration")
-> "ation"

LongestCommonEnding("potent", "tent")
-> "tent"

LongestCommonEnding("skyscraper", "carnivore")
-> ""
```

Hints

[label=-] Return an empty string if there exists no common ending.

3.7 Clear Brackets

Create a function `Brackets()` that takes a string and checks that the brackets in the math expression are correct. The function should return `true` or `false`.

Examples

```
Brackets("(a*(b-c).................)")
-> true

Brackets(")(a-b-45/7*(a-34))")
-> false

Brackets("sin(90...)+..................................cos1)")
-> false
```

Hints

- The string may not contain brackets, then return `true`.
- The string may contain spaces.
- The string may be empty.

3.8 Count the Number of Duplicate Characters

Create a function that takes a string and returns the number of alphanumeric characters that occur more than once.

Examples

```
DuplicateCount("abcde")
-> 0

DuplicateCount("aabbcde")
-> 2

DuplicateCount("Indivisibilities")
-> 2

DuplicateCount("Aa")
-> 0
// Case sensitive
```

Hints

- Duplicate characters are case sensitive.
- The input string will contain only alphanumeric characters.

3.9 Uban Numbers

A number *n* is called *uban* if its name (in English) does not contain the letter “u”. In particular, it cannot contain the terms “four”, “hundred”, and “thousand”, so the uban number following 99 is 1,000,000.
Write a function to determine if the given integer is uban.

Examples

```
IsUban(456)
-> false

IsUban(25)
-> true

IsUban(1098)
-> false
```

Hints

- Convert the integer to its English word representation. Use a library or a function to achieve this.
- Check if the word representation contains the letter “u”.
- If the letter “u” is present in the word representation, the number is not uban.
- If the letter “u” is not present, the number is uban.
- Consider edge cases like very large numbers or zero.

Functions and Function Overloading

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

4.1 Climbing Stairs

You are climbing a staircase. It takes *n* steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Specifications:

1. **Function Signature:**

```
int climbStairs(int n);
```

2. **Input:**
   - An integer *n* representing the total number of steps to reach the top.
3. **Output:**
   - An integer representing the number of distinct ways to reach the top.
4. **Constraints:**

- $1 \leq n \leq 45$

Examples:

Example 1:

```
Input: n = 2 Output:
2
Explanation: There are two ways to climb to the top.
  1. 1 step + 1 step
  2. 2 steps
```

Example 2:

```
Input: n = 3 Output:
3
Explanation: There are three ways to climb to the top.
  1. 1 step + 1 step + 1 step
  2. 1 step + 2 steps
  3. 2 steps + 1 step
```

Implement the function `climbStairs` and also provide driver code to solve the problem.

Hints:

- Consider the base cases: $n = 1$ and $n = 2$.
- Think about the recursive relationship. If you are on the nth step, you could have arrived from the $(n-1)$th step or the $(n-2)$th step.

Try

- Test the function with small values of n such as 1, 2, 3, 4, and larger values up to 45 to verify correctness and performance.

4.2 Prime Number Utility

Create a program that offers three prime-related functionalities using three different functions, invoked based on user input. Specifically:

1. **Input:** Prompt the user to select an option:
   - Check if a number is prime.
   - Find the nth prime number.
   - Display a series of prime numbers up to a given number.
2. **Function Design:**
   - Implement a function `bool isPrime(int number)` that:
     – Returns `true` if the number is prime.
     – Returns `false` otherwise.
   - Implement a function `int findNthPrime(int n)` that:
     – Uses the `isPrime` function to find and return the nth prime number.

- Implement a function `void printPrimeSeries(int limit)` that:
  - Uses the `isPrime` function to print all prime numbers up to the given limit.

3. **Output:**
   - Based on the user's selection, the program will:
     - Print "Prime" or "Not Prime" if the first option is selected.
     - Print the nth prime number if the second option is selected.
     - Print the series of prime numbers up to the given limit if the third option is selected.

Write the C# code for the Prime Number Utility program using the described functions.

Hints:

- A prime number is a number greater than 1 that has no divisors other than 1 and itself.
- Use a loop to check for divisors up to the square root of the number for efficiency.
- To find the nth prime, iterate through numbers, checking for primality, and count primes found until the nth prime is reached.
- To display a series of prime numbers, iterate through numbers up to the limit and print each prime number.

4.3 Zodiac Sign Calculator

`ZodiacSignCalculator` determines the zodiac sign based on a given birthdate using func- tions. The program should:

1. **Input**: Prompt the user to enter their birthdate (DD/MM/YYYY).
2. **Functionality**: Implement a function to determine the zodiac sign based on the entered birthdate.
3. **Output**: Display the calculated zodiac sign corresponding to the entered birthdate.

Hints:

- Use functions to encapsulate the logic for determining the zodiac sign.
- Validate the input format and range of dates to ensure accurate zodiac sign determination.

Try:

- Test the program with various birthdates to verify correct zodiac sign calculation.
- Ensure the program handles edge cases such as leap years and valid input ranges.

4.4 Days in a Month Calculator

`DaysInMonthCalculator` determines the number of days in a given month of a specific year using functions. The program should:

1. **Input**: Prompt the user to enter a month (1 for January, 2 for February, etc.) and a year.
2. **Functionality**: Implement a function to calculate and return the number of days in the specified month.
3. **Output**: Display the calculated number of days in the entered month and year.

Hints:

- Use functions to encapsulate the logic for handling leap years and month-specific day counts.
- Validate the input to ensure valid month and year ranges.

Try:

- Test the program with different months and years, including leap years, to ensure accurate day count calculation.
- Verify that the program correctly handles edge cases such as February in leap years.

4.5 Date Validator

`DateValidator` checks if a given date is valid using functions. The program should:

1. **Input**: Prompt the user to enter a date in the format (DD/MM/YYYY).
2. **Functionality**: Implement a function to validate if the entered date is valid, considering leap years and month-specific day counts.
3. **Output**: Display a message indicating whether the entered date is valid or invalid.

Hints:

- Use functions to encapsulate the logic for validating date entries.
- Ensure proper validation of day ranges based on month and leap year calculations.

Try:

- Test the program with various dates to verify accurate validation of both valid and invalid dates.
- Ensure the program handles edge cases such as February 29th in leap years.

4.6 Age Difference Calculator

Develop a program named `AgeDifferenceCalculator` that calculates the age difference between two individuals using functions. The program should:

1. **Input**: Prompt the user to enter the birthdates of two people in the format (DD/M-M/YYYY).
2. **Functionality**: Implement a function to calculate and return the difference in years, months, and days between the two birthdates.
3. **Output**: Display the calculated age difference in a readable format (e.g., “X years Y

months Z days”).

Hints

:

- Use functions to encapsulate the logic for age calculation and handling date differences.
- Validate the input to ensure accurate date formatting and valid date ranges.

Try

- Test the program with different birthdate pairs to verify accurate age difference calculation.
- Ensure the program correctly handles edge cases such as identical birthdates and reversed input order.

4.7 Elapsed Time Calculator

`ElapsedTimeCalculator` calculates the elapsed time between two timestamps within the same day using functions. The program should:

1. **Input**: Prompt the user to enter two timestamps in hours, minutes, and seconds format.
2. **Functionality**: Implement a function to calculate and return the elapsed time in hours, minutes, and seconds.
3. **Output**: Display the calculated elapsed time in a readable format (e.g., “X hours Y minutes Z seconds”).

Hints:

- Use functions to encapsulate the logic for time difference calculation and handling time rollover.
- Validate the input to ensure valid time entries (0 ¡= hours ¡ 24, 0 ¡= minutes ¡ 60, 0 ¡= seconds ¡ 60).

Try:

- Test the program with different timestamp pairs to verify accurate calculation of elapsed time.
- Ensure the program handles edge cases such as timestamps crossing over midnight or identical timestamps.

4.8 Count of Specific Weekdays

`CountSpecificWeekdays` counts the number of specific weekdays (e.g., Mondays) between two given dates using functions. The program should:

1. **Input**: Prompt the user to enter two dates in the format (DD/MM/YYYY) and specify the weekday to count.

2. **Functionality**: Implement a function to iterate through each day between the two dates and count occurrences of the specified weekday.

3. **Output**: Display the total count of the specified weekday within the date range.

Hints:

- Use functions to encapsulate the logic for date iteration and weekday comparison.

- Validate the input to ensure accurate date formatting and valid date ranges.

Try:

- Test the program with various date ranges and weekdays to verify accurate counting.

- Ensure the program handles edge cases such as spanning across different months or years.

4.9 Polymorphic Printing

Function overloading is a feature that allows multiple functions with the same name but different parameter types or numbers, enabling the same function name to perform different tasks based on how it is called.
Create a program that demonstrates function overloading in C#. Specifically, define multiple versions of functions that handle different types of calculations or tasks but share the same function name.

1. **Function Design:**

   - Define a function `print()` that prints different types of data (e.g., integer, floating point number, string) to the console.
   - Overload the `print()` function to handle different data types and formats.
   - Demonstrate the use of function overloading with at least two different versions of the `print()` function, each accepting different parameters but sharing the same name.

2. **Output:** Output the results of calling the overloaded `print()` functions with different data types to demonstrate their functionality.

Hints:

- Ensure that each overloaded function has a distinct parameter list to differentiate them during function calls.

- Implement error handling or default behavior for unexpected or unsupported data types.

Try:

- Test the program with various data types (e.g., integer, float, string) to validate the functionality of each overloaded `print()` function.

- Verify the handling of edge cases such as empty strings or invalid input values.

Motley Coding Tasks

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

5.1 Is the Phone Number Formatted Correctly?

Create a function that accepts a string and returns `true` if it's in the format of a proper phone number and `false` if it's not. Assume any number between 0-9 (in the appropriate spots) will produce a valid phone number.

Input

- A string representing the phone number.

Requirements

- The phone number must be in the format: (123) 456-7890
- Each digit can be any number between 0-9.
- There should be a space after the closing parenthesis.

Output

- Return `true` if the phone number is correctly formatted.
- Return `false` if the phone number is not correctly formatted.

Examples

1. `IsValidPhoneNumber("(123) 456-7890")` → `true`
2. `IsValidPhoneNumber("1111)555 2345")` → `false`
3. `IsValidPhoneNumber("098) 123 4567")` → `false`

Hints

- Consider using regular expressions to match the phone number format.
- Ensure that the string has exactly 14 characters.
- Check for the specific positions of characters: parentheses, space, and hyphen.

5.2 Basic E-Mail Validation

Create a function that accepts a string, checks if it's a valid email address, and returns either `true` or `false`, depending on the evaluation.

Input

- A string representing the email address to be validated.

Requirements

- The string must contain an `@` character.
- The string must contain a `.` character.

- The `@` must have at least one character in front of it.

- The `.` and the `@` must be in the appropriate places:

  – The `.` must come after the `@`.
  – There should be characters between the `@` and `.`.

Output

- Return `true` if the email address is valid.

- Return `false` if the email address is not valid.

Examples

1. `ValidateEmail("@gmail.com") → false`

2. `ValidateEmail("hello.gmail@com") → false`

3. `ValidateEmail("gmail") → false`

4. `ValidateEmail("hello@gmail") → false`

5. `ValidateEmail("hello@edabit.com") → true`

Hints

- Check the Tests tab to see exactly what's being evaluated.

- You can solve this challenge with RegEx, but it's intended to be solved with logic. Solutions using RegEx will be accepted but frowned upon.

5.3 Valid C# Comments

In C#, consider two types of comments:

- Single-line comments start with `//`

- Multi-line or block comments start with `/*` and end with `*/`

The input will be a sequence of `//`, `/*`, and `*/`. Every `/*` must have a `*/` that immediately follows it. Additionally, there can be no single-line comments in between multi-line comments.

Create a function that returns `true` if the comments are properly formatted, and `false` otherwise.

Input

- A string representing the sequence of comments.

Requirements

- Every `/*` must be followed by a `*/`.

- There must be no single-line comments (`//`) within multi-line comments (`/*` and `*/`).

Output

- Return `true` if the comments are properly formatted.
- Return `false` if the comments are not properly formatted.

Examples

1. `CommentsCorrect("//////")` → `true`
   - 3 single-line comments: `"//", "//", "//"`
2. `CommentsCorrect("/**//**////**/")` → `true`
   - 3 multi-line comments + 1 single-line comment: `"/*", "*/", "/*", "*/", "//", "/*", "*/"`
3. `CommentsCorrect("///*/**/")` → `false`
   - The first `/*` is missing a `*/`
4. `CommentsCorrect("/////")` → `false`
   - The 5th `/` is single, not a double `//`

5.4 IPv4 Validation

Problem Description

Create a function that takes a string (an IPv4 address in standard dot-decimal format) and returns `true` if the IP is valid, or `false` if it's not.

Input

- A string representing the IPv4 address.

Requirements

- The IPv4 address must be in the standard dot-decimal format: `X.X.X.X`, where `X` is a number.
- Each `X` must be a number between 0 and 255.
- Each `X` must not have leading zeros (e.g., `045` is invalid).
- The last digit may not be zero (e.g., `1.2.3.0` is invalid).
- IPv6 addresses are not valid.
- The address must contain exactly four octets.

Output

- Return `true` if the IPv4 address is valid.
- Return `false` if the IPv4 address is not valid.

Examples

1. `IsValidIP("1.2.3.4")` → `true`
2. `IsValidIP("1.2.3")` → `false`
3. `IsValidIP("1.2.3.4.5")` → `false`
4. `IsValidIP("123.45.67.89")` → `true`
5. `IsValidIP("123.456.78.90")` → `false`
6. `IsValidIP("123.045.067.089")` → `false`

Hints

- IPv6 addresses are not valid.
- Leading zeros are not valid (e.g., `123.045.067.089` should return false).
- Numbers must be between 1 and 255.
- The last digit must not be zero, but any other octet may have a zero.

5.5 Password Validation

Create a function that validates a password based on the following rules:

- Length must be between 6 and 24 characters.
- Must contain at least one uppercase letter (A-Z).
- Must contain at least one lowercase letter (a-z).
- Must contain at least one digit (0-9).
- Maximum of 2 repeated characters.

- Supported special characters include:

`! @ # $ % &̂ * ( ) + = _ - { } [ ] : ; " ' ? < > , .`

Create a function that returns `true` if the password is valid and `false` otherwise.

Input

- A string representing the password to be validated.

Requirements

- The password must be between 6 and 24 characters long.
- It must contain at least one uppercase letter.
- It must contain at least one lowercase letter.
- It must contain at least one digit.
- The password can have a maximum of 2 repeated characters.
- Special characters are supported but not required.

Output

- Return `true` if the password meets all the requirements.
- Return `false` if the password does not meet the requirements.

Examples

1. `ValidatePassword("P1zz@")` → `false`
   – Too short.
2. `ValidatePassword("iLoveYou")` → `false`
   – Missing a number.
3. `ValidatePassword("Fhg93@")` → `true`
   – OK!

Hints

- Ensure the password meets all the length, character, and repetition requirements.
- Special characters are optional but valid.

5.6 Valid Hex Code

Create a function that determines whether a string is a valid hex code.

A hex code must begin with a pound key # and is exactly 6 characters in length. Each character must be a digit from 0-9 or an alphabetic character from A-F. All alphabetic characters may be uppercase or lowercase.

Examples

```
IsValidHexCode("#CD5C5C")
-> true

IsValidHexCode("#EAECEE")
-> true

IsValidHexCode("#eaecee")
-> true

IsValidHexCode("#CD5C58C")
-> false
// Length exceeds 6

IsValidHexCode("#CD5C5Z")
-> false
// Not all alphabetic characters in A-F

IsValidHexCode("#CD5C&C")
-> false
// Contains unacceptable character

IsValidHexCode("CD5C5C")
-> false
// Missing #
```

Hints

- Check if the string starts with a pound key (#).
- Verify that the length of the string is exactly 7 characters (including the pound key).
- Ensure that all characters following the pound key are either digits (0-9) or letters in the range A-F (case insensitive).
- Use string methods or regular expressions to validate the characters.
- Consider converting the string to uppercase for a uniform comparison.

5.7 Retrieve Host Name

Create a function that retrieves the host name of the local machine. The function should return the host name of the machine on which the code is running.

Input

- No input is required for this function.

Requirements

- The function should retrieve the host name of the local machine.
- The function should return the host name as a string.

Output

- Return a string containing the host name of the local machine.

Examples

1. `GetLocalHostName() → "MyComputerName"`

Hints

- The host name is typically the name assigned to the computer by the operating system.
- Use the `Dns.GetHostName()` method to retrieve the host name.

5.8 Retrieve Host Name and IP Address

Create a function that retrieves the host name and IP addresses of the local machine. The function should display the host name and all associated IP addresses.

Input

- No input is required for this function.

Requirements

- The function should retrieve the host name of the local machine.
- The function should retrieve all associated IP addresses of the local machine.
- The function should display the host name and each IP address on a new line.

Output

- Display the host name and all associated IP addresses of the local machine.

Examples

```
Host Name of machine = MyComputerName
IP Address of Machine is
192.168.1.2
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

Hints

- The host name is typically the name assigned to the computer by the operating system.
- The IP addresses can be both IPv4 and IPv6.
- Use the `Dns.GetHostName()` method to retrieve the host name.
- Use the `Dns.GetHostAddresses(string)` method to retrieve the IP addresses.
- Import necessary packages: `System`, `System.Net`.

5.9 Retrieve IP Address of a Domain Name

Create a function that retrieves the IP addresses of a given domain name. The function should display the IP addresses associated with the provided domain name.

Input

- A string `domainName` representing the domain name to be resolved.

Requirements

- The function should retrieve all IP addresses associated with the given domain name.
- The function should display each IP address on a new line.
- Handle any potential exceptions gracefully and provide an appropriate error message.

Output

- Display the IP addresses associated with the provided domain name.

Examples

```
GetIPAddresses("www.bing.com");

Output:
IPAddress of www.bing.com is
13.107.21.200
204.79.197.200
```

Hints

- The domain name should be a valid domain name string.
- The IP addresses can be both IPv4 and IPv6.
- Use the `Dns.GetHostAddresses(string)` method to retrieve the IP addresses.
- Import necessary packages: `System`, `System.Net`.
- Handle exceptions using a try-catch block to ensure graceful error handling.

5.10 Retrieve IPv4 and IPv6 Addresses - Check Address Family

Create a function that retrieves and displays both IPv4 and IPv6 addresses of the local machine separately by checking the Address Family. The function should first output the host name of the machine, followed by separate lists of IPv4 and IPv6 addresses.

Input

- No input is required for this function.

Requirements

- The function should retrieve the host name of the local machine using `Dns.GetHostName()`.
- The function should retrieve all associated IP addresses of the local machine using `Dns.GetHostAddress`
- The function should filter and display IPv4 addresses separately.
- The function should filter and display IPv6 addresses separately.
- Display the host name followed by the lists of IPv4 and IPv6 addresses.

Output

- Display the host name of the local machine.
- Display the IPv4 addresses of the local machine, each on a new line.
- Display the IPv6 addresses of the local machine, each on a new line.

Examples

```
Host Name of machine = MyComputerName
IPv4 of Machine is
192.168.1.2
IPv6 of Machine is
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

Hints

- Ensure that you filter IP addresses by `AddressFamily` to distinguish between IPv4 and IPv6 addresses.
- The function should handle both types of addresses and display them clearly.
- Import the necessary packages: `System`, `System.Net`.
- Use the `Dns.GetHostName()` method to get the host name.
- Use the `Dns.GetHostAddresses(string)` method to get all IP addresses associated with the host name.
- Use LINQ to filter IP addresses based on `AddressFamily` to separate IPv4 and IPv6 addresses.

5.11 Retrieve Host Name Based on IP Address

Create a function that takes an IP address as input and returns the host name associated with that IP address. The function should use the `Dns.GetHostEntry()` method to obtain the host information and then extract and display the host name.

Input

- A string representing an IP address (e.g., "`204.79.197.200`").

Requirements

- The function should use the `Dns.GetHostEntry(string)` method to retrieve the `IPHostEntry` for the given IP address.
- Extract the host name from the `IPHostEntry` object.
- Display the host name associated with the given IP address.

Output

- Display the host name of the IP address.

Examples

```
Input: "204.79.197.200"
Output: "example.microsoft.com"
```

Hints

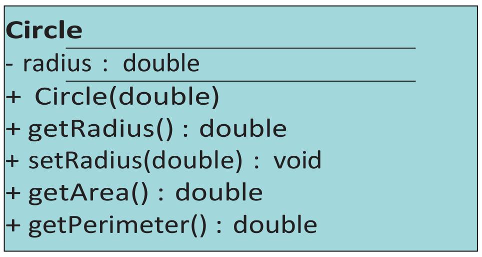
- Ensure that the IP address provided is valid and reachable.
- The host name may not always be resolvable depending on network configurations and the validity of the IP address.
- Import the necessary packages: `System`, `System.Net`.
- Use the `Dns.GetHostEntry(string)` method to retrieve the `IPHostEntry` for the given IP address.
- Access the `HostName` property of the `IPHostEntry` object to get the host name.

Classes & Objects

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

6.1 Make a Circle with OOP

Your task is to create a `Circle` constructor that creates a circle with a radius provided by an argument. The circles constructed must have two getters `GetArea()` ($\pi \cdot r^2$) and `GetPerimeter()` ($2 \cdot \pi \cdot r$) which give both respective areas and perimeter (circumference).



Examples

```
Circle circy = new Circle(11);
circy.getArea();

// Should return 380.132711084365.

Circle circy = new Circle(4.44);
circy.getPerimeter();

// Should return 27.897342763877365
```

Hints

- Don't worry about floating point precision - the solution should be accurate enough.
- This is more of a tutorial than a challenge, so the topic covered may be considered ad- vanced.

6.2 Fruit Smoothie

Create a class `Smoothie` and do the following:

- Create a property called `Ingredients`.
- Create a `GetCost` method which calculates the total cost of the ingredients used to make the smoothie.
- Create a `GetPrice` method which returns the cost of the smoothie plus 50% markup. Round to two decimal places.
- Create a `GetName` method which gets the ingredients and puts them in alphabetical order into a nice descriptive sentence:
  - If there are multiple ingredients, add the word "Fusion" to the end.
  - If there is only one ingredient, add "Smoothie".
  - Change "-berries" to "-berry" for individual berry names.

Smoothie
- Ingredients : string[]
+ Smoothie(string[])
+ GetCost() : string
+ GetPrice() : string
+ GetName() : string

Ingredient Prices:

Ingredient	Price
Strawberries	1.50
Banana	0.50

Mango	2.50
Blueberries	1.00
Raspberries	1.00
Apple	1.75
Pineapple	3.50

Examples:

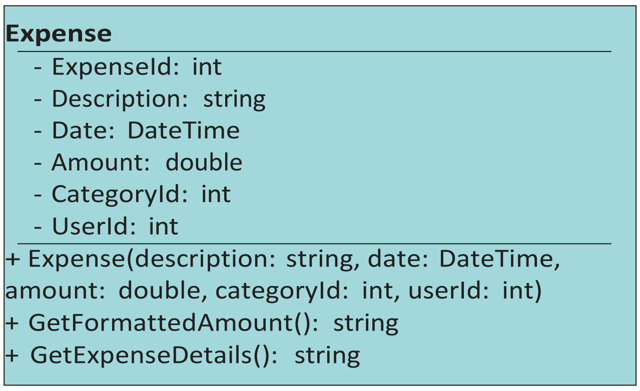
```
s1 = Smoothie(new string[] {
"Banana" }) s1.Ingredients -> {
"Banana" } s1.GetCost() ->
"£0.50"
s1.GetPrice() -> "£1.25"
s1.GetName() -> "Banana
Smoothie"

s2 = Smoothie(new string[] { "Raspberries", "Strawberries",
"Blueberries" }) s2.ingredients -> { "Raspberries",
"Strawberries", "Blueberries" } s2.GetCost() -> "£3.50"
s2.GetPrice() -> "£8.75"
s2.GetName() -> "Blueberry Raspberry Strawberry Fusion"
```

Hints:

Feel free to check out the Resources tab for a refresher on classes.

6.3 Expense Tracker

Design and implement an `Expense` class to manage individual expenses within an Expense Tracker application. The class should provide functionalities to log, retrieve, and display expense details effectively.



```
Expense
  - ExpenseId: int
  - Description: string
  - Date: DateTime
  - Amount: double
  - CategoryId: int
  - UserId: int
+ Expense(description: string, date: DateTime,
amount: double, categoryId: int, userId: int)
+ GetFormattedAmount(): string
+ GetExpenseDetails(): string
```

Output:

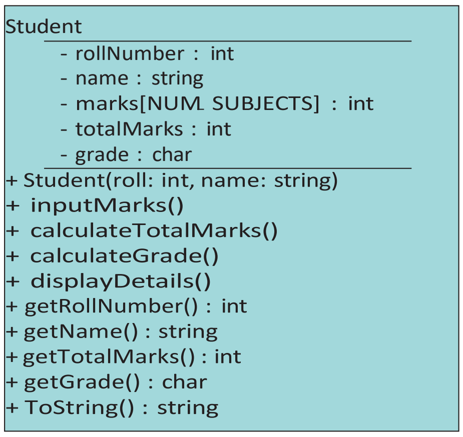
- `Constructor`: No direct output. Initializes the `Expense` object with the provided data.
- `GetFormattedAmount()`: Returns a formatted string representing the monetary amount of the expense.
- `GetExpenseDetails()`: Returns a formatted string with detailed information about the expense.

Hints:

- **Encapsulation**: Ensure attributes are properly encapsulated (private or protected access modifiers) to maintain data integrity.
- **Date Formatting**: Use appropriate formatting methods to display the date in a readable format.
- **String Formatting**: Utilize C#'s string interpolation or formatting methods (`ToString()`, format specifiers) for consistent and clear output.

6.4 Student Grades

The Student Grades class is designed to manage student details including roll number, name, and marks in multiple subjects. It calculates total marks and grades for each student and provides functionalities to input student details, process marks, calculate grades, and display student information.



```
Student
    - rollNumber : int
    - name : string
    - marks[NUM_SUBJECTS] : int
    - totalMarks : int
    - grade : char
+ Student(roll: int, name: string)
+ inputMarks()
+ calculateTotalMarks()
+ calculateGrade()
+ displayDetails()
+ getRollNumber() : int
+ getName() : string
+ getTotalMarks() : int
+ getGrade() : char
+ ToString() : string
```

Inputs

- Roll number and name for each student.
- Marks in multiple subjects (e.g., Math, Science, English) for each student.

Process Requirements

- Initialize student details and marks using the constructor.
- Input marks for each subject using the `inputMarks()` method.
- Calculate total marks using the `calculateTotalMarks()` method.
- Calculate the grade based on total marks using the `calculateGrade()` method.
- Display student details including roll number, name, marks in each subject, total marks, and grade using the `displayDetails()` method.

Output

- Display student details including roll number, name, marks in each subject, total marks, and grade.

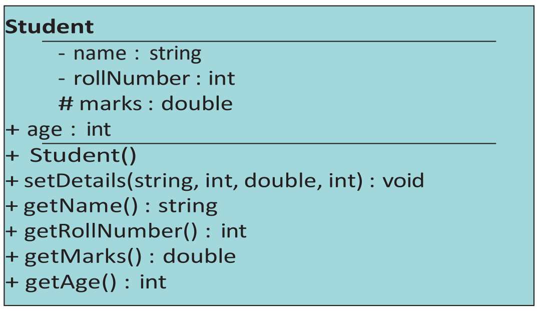
Hints

Grades can be assigned based on the total marks as follows:

- **A:** 90 - 100
- **B:** 80 - 89
- **C:** 70 - 79
- **D:** 60 - 69
    - **F:** below 60

6.5 Control Your Code: Access Specifiers

Data hiding is a key feature of Object-Oriented Programming that restricts direct access to the internal details of a class. This prevents functions from directly manipulating the internal state of a class type. Access to class members is controlled using public, private, and protected labels within the class definition. These labels, known as access specifiers, dictate the level of access permitted for the class members.
Demonstrate the concepts of data hiding and access specifiers in a class `Student`. The `Student` class should have private, protected, and public data members. Implement member functions to set and get these details, ensuring that the internal representation of the class type is not directly accessible outside the class.



```
Student
    - name : string
    - rollNumber : int
    # marks : double
+ age : int
+ Student()
+ setDetails(string, int, double, int) : void
+ getName() : string
+ getRollNumber() : int
+ getMarks() : double
+ getAge() : int
```

The `Student` class encapsulates the properties of a student while ensuring that the internal data (name, roll number, marks) is hidden from direct access outside the class. This is achieved using access specifiers: private, protected, and public.

Inputs

Details (`name`, `rollNumber`, `marks`, `age`) for a student.

Process Requirements

1. Define the `Student` class with:

Private data members: `name`, `rollNumber`.

Protected data member: `marks`.

Public data member: `age`.

2. Implement public member functions `setDetails`, `getName`, `getRollNumber`, `getMarks`, and `getAge` to set and retrieve the values of the data members.

3. Ensure that the private data members cannot be accessed directly outside the class.

Output

- Print the details (`name`, `rollNumber`, `marks`, `age`) of the student using the public member functions.

Hints

- Use private access specifier to hide sensitive data members.

- Use protected access specifier for data members that should be accessible in derived classes.

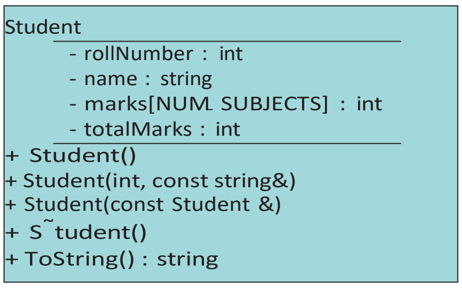
- Use public access specifier for data members that can be accessed directly.
- Example usage of the Student class with different access specifiers:

6.6 Craft Constructors

Constructors are special member functions responsible for initializing objects of a class. They are invoked automatically when an object is created. Constructors can initialize data members, allocate resources, and set default values based on parameters provided during object creation. They are defined with the same name as the class and can be overloaded to accept different parameter sets.

Destructors are also special member functions, that are invoked automatically when an ob- ject goes out of scope or is explicitly deleted. They perform cleanup tasks such as releasing resources (like memory or handles) acquired by the object during its lifetime. Destructors are named with a tilde ( ) followed by the class name and cannot have parameters or return types. They are essential for managing dynamic memory and ensuring proper resource deallocation.

Implement constructors for default initialization, parameterized initialization, copy opera- tion, and a destructor to manage resources for the following class



```
Student
    - rollNumber : int
    - name : string
    - marks[NUM SUBJECTS] : int
    - totalMarks : int
+ Student()
+ Student(int, const string&)
+ Student(const Student &)
+ S˜tudent()
+ ToString() : string
```

Input:

- **rollNumber**: Integer value representing the student's roll number.
- **name**: String representing the student's name.
- **marks[NUM SUBJECTS]**: Array of integers representing marks in various subjects.

Process Requirements:

- Implement a default constructor to initialize the 'Student' object with default values.
- Create a parameterized constructor to set initial values for 'rollNumber' and 'name'.
- Define a copy constructor to copy data from another 'Student' object.
- Implement a destructor to clean up resources when the object is destroyed.

Output:

- Display methods to show student details including 'rollNumber', 'name', 'marks', and 'totalMarks'.

Hints:

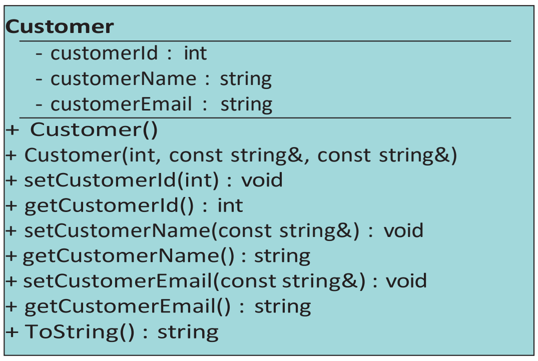
```
// Create Student objects using different constructors

// Display student details

// Clean up resources
// Destructor automatically called when objects go out of scope
```

Try:

- Use initialization lists and proper memory management techniques.
- Ensure proper handling of dynamic memory if applicable.

6.7 Clearing Up Ambiguity: The Role of this Pointer

A special pointer, 'this' that refers to the current object instance within non-static member functions. It allows these functions to access and manipulate the object's data members and call other member functions of the same object.
Design a `Customer` class to manage customer information in a retail application. The class should include:



```
Customer
  - customerId : int
  - customerName : string
  - customerEmail : string
+ Customer()
+ Customer(int, const string&, const string&)
+ setCustomerId(int) : void
+ getCustomerId() : int
+ setCustomerName(const string&) : void
+ getCustomerName() : string
+ setCustomerEmail(const string&) : void
+ getCustomerEmail() : string
+ ToString() : string
```

Input

- **Customer Information:** Inputs for customer details including `customerId`, `customerName`, and `customerEmail`.

Process

Implement a `Customer` class with:

- Private member variables: `customerId` (int), `customerName` (string), `customerEmail` (string).
- Constructors: Default constructor and parameterized constructor using the `this` pointer for initialization.
- Methods:
  * Setters and getters for each member variable, demonstrating **self-reference** us- ing the `this` pointer within methods.
  * Resolve **disambiguation** between instance variables and parameters/local vari- ables with the same name using the `this` pointer.
  * Enable **method chaining** by returning the `this` pointer from appropriate meth- ods.
  * Illustrate **passing object as argument** by implementing a method that accepts another `Customer` object.
  * Show how to **return the current object** (`this`) from methods to maintain method chaining or return the object state.

Output

- Display customer details including `customerId`, `customerName`, and `customerEmail` using the implemented methods.

Hints

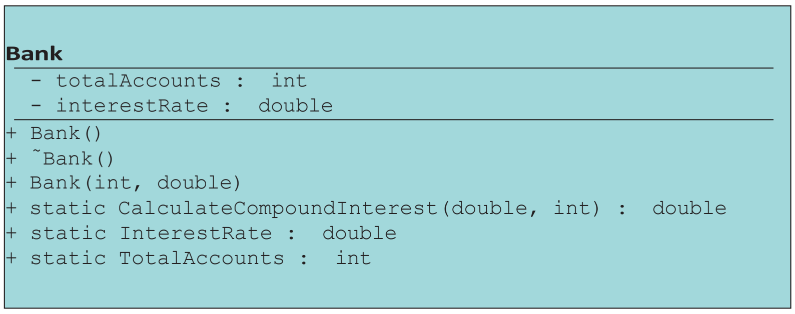
- Use the `this` pointer to reference the current object within its own methods and con- structors.
- Demonstrate how the `this` pointer resolves naming conflicts and ensures clarity in variable usage.
- Showcase the versatility of the `this` pointer in enabling efficient method chaining and object-oriented practices.

Try

- Create instances of the `Customer` class and demonstrate setting customer details using method chaining.
- Call methods sequentially on the same object to observe method chaining in action.
- Pass a `Customer` object as an argument to another method to illustrate object passing.
- Return the current object from a method and use it in a method chain to verify function- ality.

6.8 Consistent Behavior Across Objects

Demonstrate the usage of various `static` members, including static fields, properties, methods, and a static constructor. Explore their behaviors and interactions within the context of a simple application.



```
Bank
  - totalAccounts : int
  - interestRate : double
+ Bank()
+ ˜Bank()
+ Bank(int, double)
+ static CalculateCompoundInterest(double, int) : double
+ static InterestRate : double
+ static TotalAccounts : int
```

Requirements

1. **Static Fields**:
   - Implement a static field to maintain a count of instances created for the class.
   - Display the current count of instances whenever a new instance is created.
2. **Static Properties**:
   - Define a static property to store and retrieve a global configuration value.
   - Ensure the property is accessible and modifiable from different parts of the application.
3. **Static Methods**:
   - Create a static method to perform a common utility task (e.g., calculating compound interest).
   - Demonstrate calling this method without creating an instance of the class.

4. **Static Constructor**:
   - Implement a static constructor to initialize any static data or perform setup tasks.
   - Print a message confirming the static constructor's execution when the application starts.
5. **Application Interaction**:
   - Create instances of the class to verify the behavior of static fields.
   - Access and modify the static property to demonstrate its global accessibility.
   - Invoke the static method to perform a utility task.
   - Observe the sequence of static constructor execution.

Example Output

Upon running the application, the output should demonstrate:

- Creation of instances and display of instance count.
- Setting and retrieving global configuration values using the static property.
- Execution of the static method to perform a task.
- Confirmation message from the static constructor indicating initialization.

Hints

- Use the `static` keyword appropriately for fields, properties, methods, and constructors.
- Consider thread safety considerations when modifying shared static data.
- Utilize console output or logging to track execution flow and demonstrate expected behavior.

Try

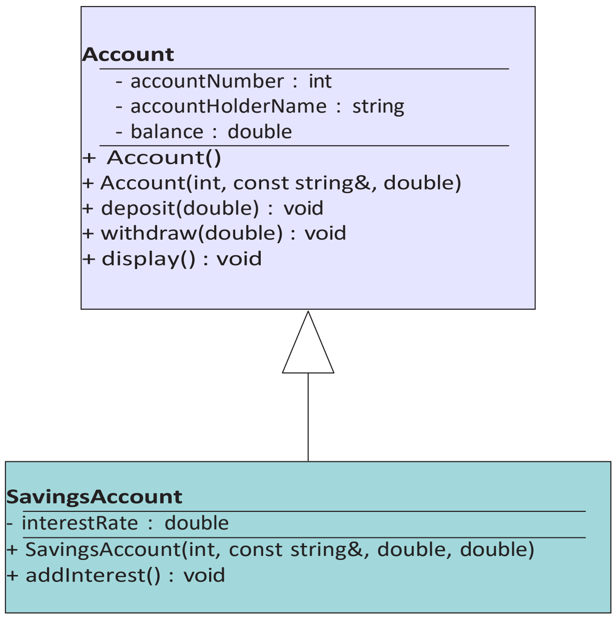
Implement the `Main` method to instantiate the `Bank` class, access static members, and verify the outputs as per the requirements.

Inheritance

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

7.1 Single Inheritance

You are tasked with designing a program to model a simple banking system using single inheritance. The program will involve creating a base class `Account` and a derived class `SavingsAccount`.



Input

- Initialize `Account` and `SavingsAccount` objects with appropriate data.
- Perform deposit and withdrawal operations on `SavingsAccount`.
- Calculate and add interest to `SavingsAccount`.

Process Requirements

1. **Account Class**:
   - Define the `Account` class with data members `accountNumber`, `accountHolderName`, and `balance`.

- Implement constructors for initializing the account.
- Methods include `deposit`, `withdraw`, and `display` to perform operations and display account details.

2. **SavingsAccount Class (Derived from Account)**:

   - Derive `SavingsAccount` from `Account`.
   - Add an additional data member `interestRate`.
   - Implement a constructor to initialize `SavingsAccount` with all parameters includ- ing `interestRate`.
   - Implement `addInterest` method to calculate and add interest to the balance based on the `interestRate`.

Output

- Display account details after each operation (deposit, withdrawal, adding interest).

Hints

- Use inheritance (`public`) to inherit `Account` properties into `SavingsAccount`.
- Implement methods to ensure proper encapsulation and validation of account operations.
- Test the program with multiple scenarios to validate the inheritance and functionality.

7.2 Multiple Inheritance

Develop a bank application using multiple inheritance to manage various types of accounts. Implement two base classes, `Account` and `Interest`, and derive a `SavingsAccount` class from both. This structure will allow efficient management of basic account functionalities and interest calculations.

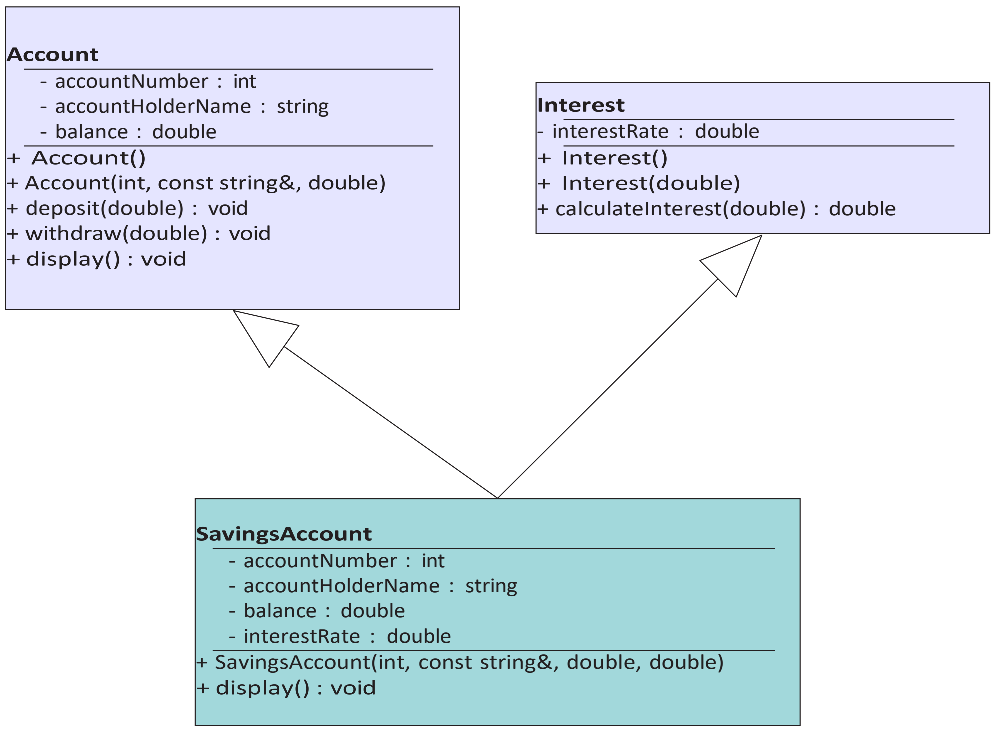
Input

- Account details: Account number, account holder name, initial balance.
- Interest rate for calculating interest.

Process Requirements

- Define the `Account` class with data members and member functions for basic account operations.
- Define the `Interest` class with data members and member functions for interest-related calculations.
- Implement the `SavingsAccount` class inheriting from both `Account` and `Interest`, integrating functionalities from both base classes.

- Implement constructors, including default and parameterized, and appropriate member functions in each class.



Output

Display account details and perform operations such as deposits, withdrawals, and interest calculations for savings accounts.

Hints

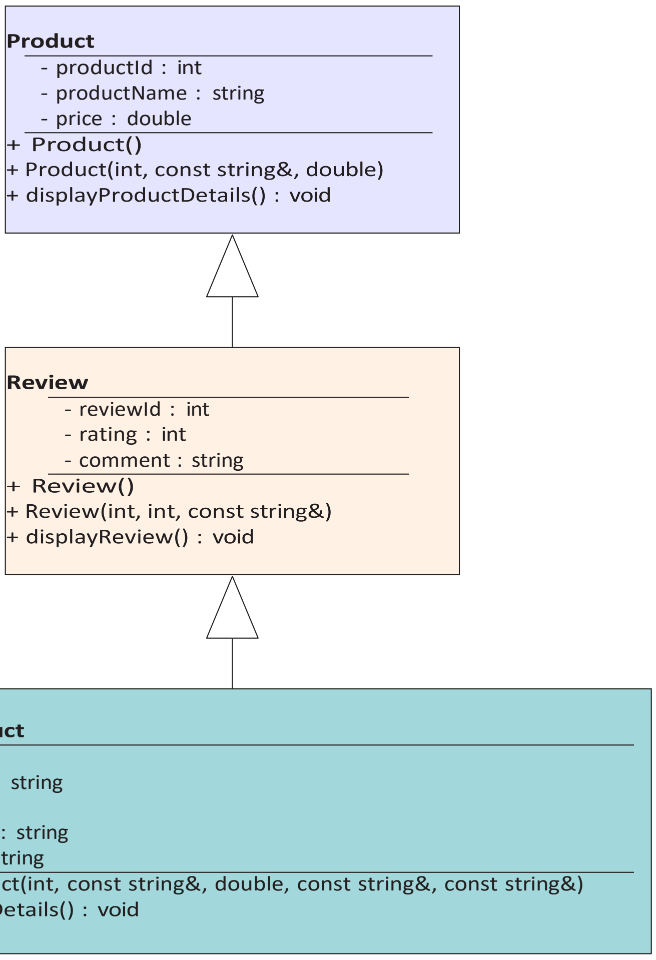
```
// Define base class Account
// Define base class Interest
// Define derived class SavingsAccount

// Usage example
SavingsAccount sa(1001, "Rama Krishna", 5000.0, 5.0);
sa.deposit(1000.0);
sa.display();
double interest = sa.calculateInterest(sa.balance); cout
<< "Interest Earned: $" << interest << endl;
```

Try

Enhance the application by adding support for different types of accounts (e.g., checking accounts, fixed deposit accounts) with specific functionalities and interest calculations.

7.3 Multi Level Inheritance

Develop an online shopping application that utilizes multi-level inheritance to manage different types of products efficiently. Implement two base classes, `Product` and `Review`, and derive an `ElectronicsProduct` class from both. This structure allows for comprehensive management of product information and customer reviews within a single class hierarchy.



Input

- Product details: product ID, product name, price.
- Review details: review ID, rating, comment.
- Additional details for electronics products: model number, brand name.

Process Requirements

- Define the `Product` class with data members and member functions for managing basic product information.
- Define the `Review` class with data members and member functions for handling customer reviews.
- Implement the `ElectronicsProduct` class inheriting from both `Product` and `Review`, integrating functionalities from both base classes.
- Implement constructors, including default and parameterized, and appropriate member functions in each class.

Output

Display product details, customer reviews, and specific information for electronics products.

Hints

```
// Define base class Product
// Define base class Review
// Define derived class ElectronicsProduct

// Usage example
ElectronicsProduct ep(1001, "Laptop", 1500.0, "Model X", "Brand Y");
ep.displayProductDetails();
ep.displayReview();
```

Try

Enhance the application by adding support for different types of products (e.g., clothing, books) with specific functionalities and review management.

7.4 Hierarchical Inheritance

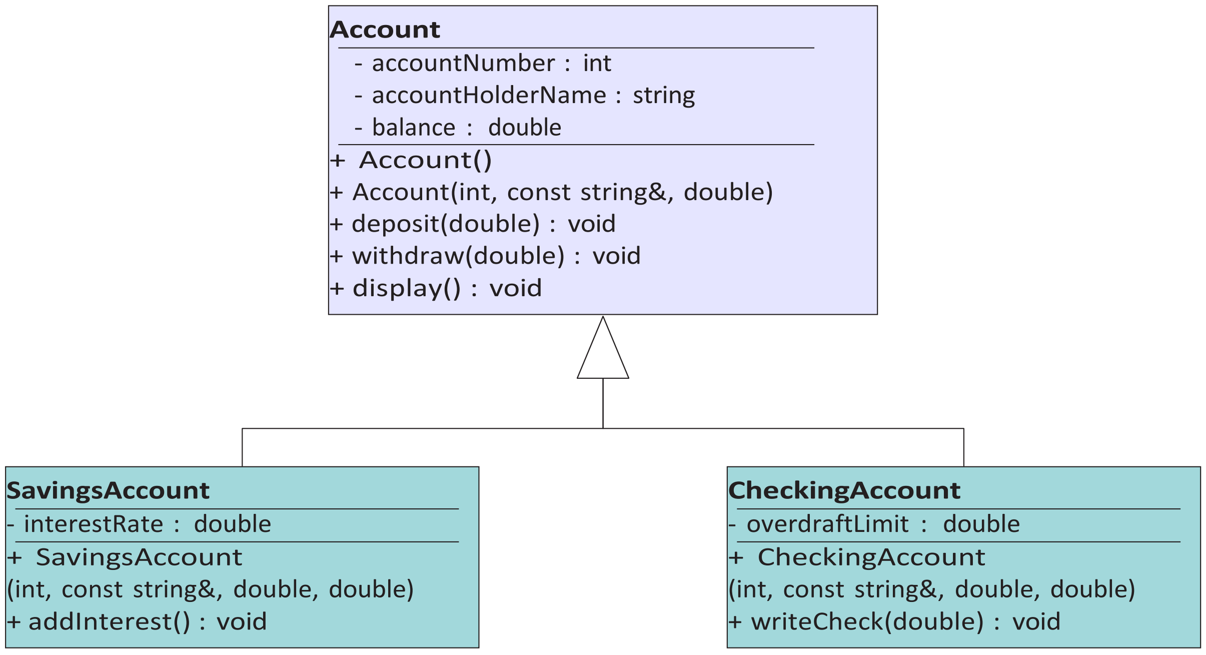
Develop a bank application using hierarchical inheritance to manage various types of accounts and related operations. Implement a base class, `Account`, and derive two classes, `SavingsAccount` and `CheckingAccount`, from it. This structure will allow efficient management of basic ac- count functionalities and specific operations for different account types.

Input

- Account details: Account number, account holder name, initial balance.
- Interest rate for savings accounts.
- Overdraft limit for checking accounts.

Process Requirements

- Define the `Account` class with data members and member functions for basic account operations.
- Implement the `SavingsAccount` class inheriting from `Account`, with additional data members and member functions for interest-related operations.
- Implement the `CheckingAccount` class inheriting from `Account`, with additional data members and member functions for check-writing operations.
- Implement constructors, including default and parameterized, and appropriate member functions in each class.


Account
- accountNumber : int
- accountHolderName : string
- balance : double
+ Account()
+ Account(int, const string&, double)
+ deposit(double) : void
+ withdraw(double) : void
+ display() : void
SavingsAccount
- interestRate : double
+ SavingsAccount (int, const string&, double, double)
+ addInterest() : void
CheckingAccount
- overdraftLimit : double
+ CheckingAccount (int, const string&, double, double)
+ writeCheck(double) : void

Output

Display account details and perform operations such as deposits, withdrawals, interest calculations for savings accounts, and check writing for checking accounts.

Hints

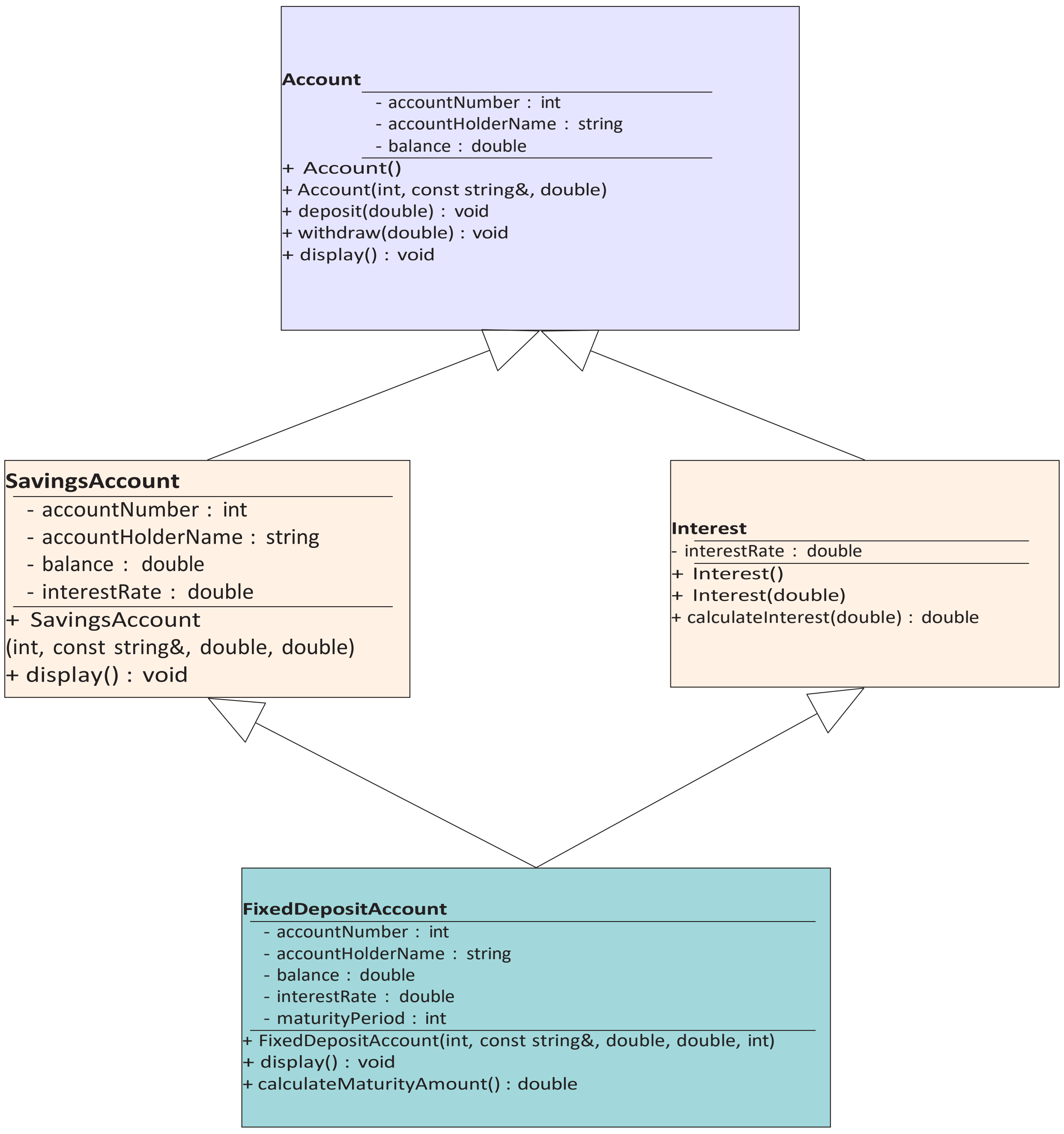
```
// Define base class Account
// Define derived class SavingsAccount
// Define derived class CheckingAccount

// Usage example
SavingsAccount sa(1001, "Rama Krishna", 5000.0, 5.0);
sa.deposit(1000.0);
sa.display();
sa.addInterest();
sa.display();

CheckingAccount ca(2001, "Sita Devi", 3000.0, 1000.0);
ca.deposit(500.0);
ca.display();
ca.writeCheck(2000.0);
ca.display();
```

Try

Enhance the application by adding support for different types of accounts (e.g., fixed deposit accounts, recurring deposit accounts) with specific functionalities and interest calculations.

7.5 Hybrid Inheritance

Develop a bank application using hybrid inheritance to manage various types of accounts and related operations. Implement two base classes, `Account` and `Interest`, and derive two classes, `SavingsAccount` and `FixedDepositAccount`, from these. This structure will allow efficient management of basic account functionalities and interest calculations.



Input

- Account details: Account number, account holder name, initial balance.
- Interest rate for calculating interest.
- Maturity period for fixed deposit accounts.

Process Requirements

- Define the `Account` class with data members and member functions for basic account operations.

- Define the `Interest` class with data members and member functions for interest-related calculations.
- Implement the `SavingsAccount` class inheriting from both `Account` and `Interest`, integrating functionalities from both base classes.
- Implement the `FixedDepositAccount` class inheriting from both `Account` and `Interest`, integrating functionalities from both base classes.
- Implement constructors, including default and parameterized, and appropriate member functions in each class.

Output

Display account details and perform operations such as deposits, withdrawals, and interest calculations for savings accounts and fixed deposit accounts.

Hints

```
// Define base class Account
// Define base class Interest
// Define derived class SavingsAccount
// Define derived class FixedDepositAccount

// Usage example
SavingsAccount sa(1001, "Rama Krishna", 5000.0, 5.0);
sa.deposit(1000.0);
sa.display();
double interest = sa.calculateInterest(sa.balance); cout
<< "Interest Earned: $" << interest << endl;

FixedDepositAccount fda(2001, "Sita Devi", 10000.0, 6.5, 5);
fda.display();
double maturityAmount = fda.calculateMaturityAmount(); cout
<< "Maturity Amount: $" << maturityAmount << endl;
```

Try

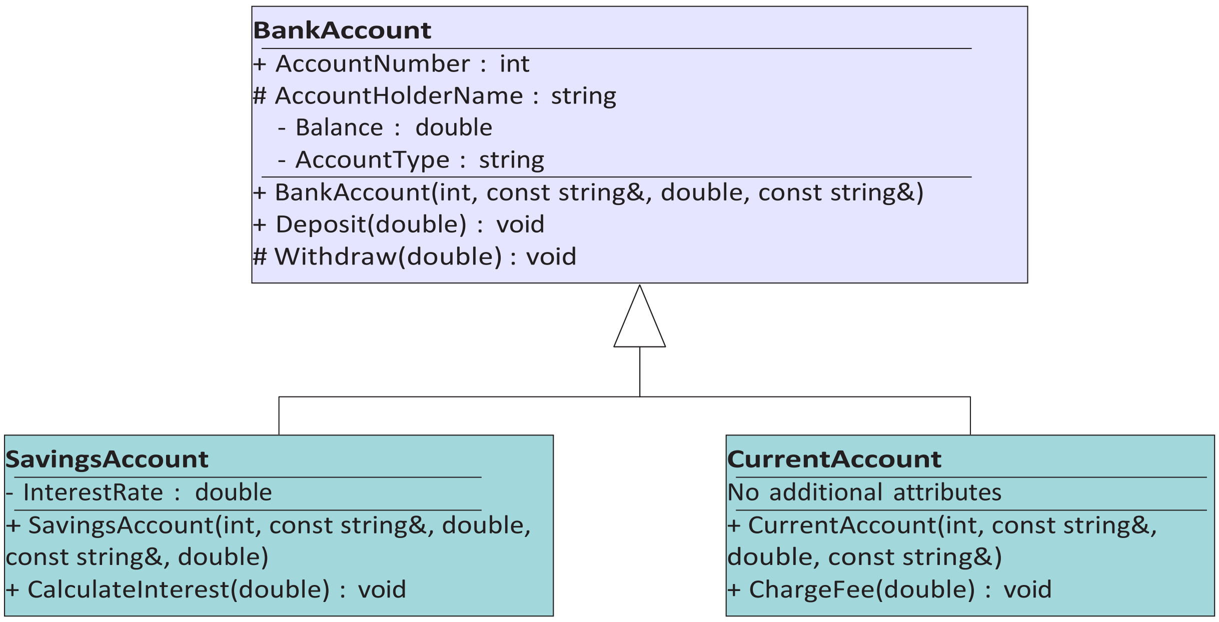
Enhance the application by adding support for different types of accounts (e.g., checking accounts, recurring deposit accounts) with specific functionalities and interest calculations.

7.6 Access Modifiers & Inheritance

Design and implement a bank application in C# to explore how different access modifiers (`public`, `protected`, `private`, etc.) affect inheritance. This exercise will enhance your understanding of object-oriented principles related to inheritance and access control in C#.

Input

- Bank account details such as account number, account holder's name, initial balance, and account type.
- Interest rate for savings accounts.
- Fees for current accounts.


BankAccount
+ AccountNumber : int
# AccountHolderName : string
- Balance : double
- AccountType : string
+ BankAccount(int, const string&, double, const string&)
+ Deposit(double) : void
# Withdraw(double) : void
SavingsAccount
- InterestRate : double
+ SavingsAccount(int, const string&, double, const string&, double)
+ CalculateInterest(double) : void
CurrentAccount
No additional attributes
+ CurrentAccount(int, const string&, double, const string&)
+ ChargeFee(double) : void

Process

- Demonstrate the usage of different access modifiers (public, protected, private) in the base and derived classes.
- Show how protected members in the base class (Balance) are accessible to derived classes (SavingsAccount and CurrentAccount) but not accessible outside of them.
- Implement functionality to deposit, withdraw, calculate interest, and charge fees based on the account type.

Output

- Display the account details including account number, account holder's name, account type, and balance after performing deposit, withdrawal, interest calculation, and fee deduction operations.

Hints

- Use access modifiers appropriately to control the accessibility of members in base and derived classes.
- Use constructors in derived classes to initialize base class data members.
- Use protected members in the base class to facilitate behavior in derived classes while keeping them encapsulated from external access.

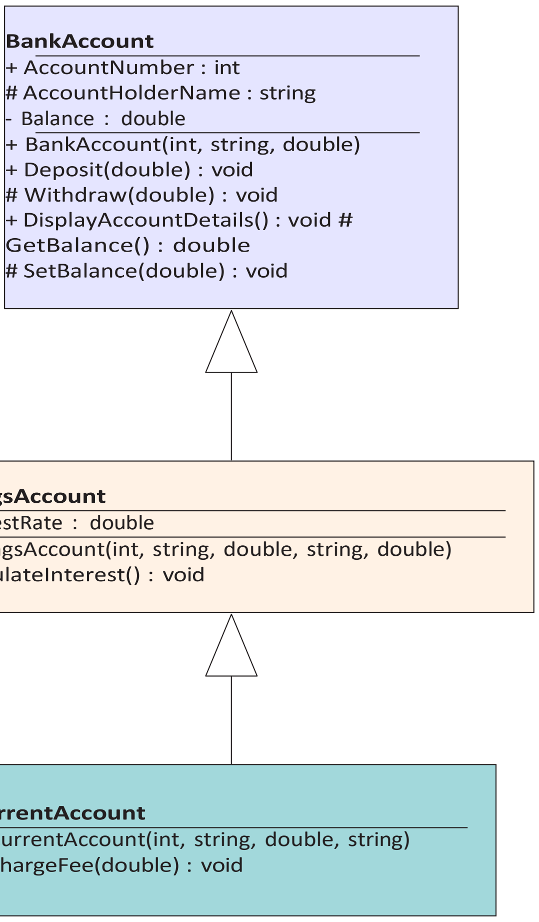
Try

- Extend the application by adding more account types (e.g., `FixedDepositAccount`) with specific functionalities and explore how access modifiers affect these new derived classes.

- Implement additional features such as account transfer and balance inquiry, ensuring proper access control using appropriate access modifiers.

7.7 Casting and Type Checking

Design and implement a bank application in C# to explore the use of the `is`, `as`, and `typeof` operators for checking and casting types in an inheritance hierarchy. This exercise will enhance your understanding of type checking and casting in object-oriented programming with C#.


BankAccount
+ AccountNumber : int
# AccountHolderName : string
- Balance : double
+ BankAccount(int, string, double)
+ Deposit(double) : void
# Withdraw(double) : void
+ DisplayAccountDetails() : void #
GetBalance() : double
# SetBalance(double) : void
SavingsAccount
- InterestRate : double
+ SavingsAccount(int, string, double, string, double)
+ CalculateInterest() : void
CurrentAccount
+ CurrentAccount(int, string, double, string)
+ ChargeFee(double) : void

Input

- Bank account details such as account number, account holder's name, initial balance, account type, and specific attributes for derived account types (e.g., interest rate for savings accounts).
- Operations to perform on different account types, such as deposit, withdraw, calculate interest, and charge fees.

Process

1. Create a base class `BankAccount` with common attributes and methods:
   - `AccountNumber` (public): a unique identifier for the account.
   - `AccountHolderName` (protected): the name of the account holder.
   - `Balance` (private): the current balance of the account.

- Constructor to initialize `AccountNumber`, `AccountHolderName`, and `Balance`.
- Methods:
  - `Deposit(double amount)` (public): allows depositing money into the account.
  - `Withdraw(double amount)` (protected): allows withdrawing money from the account.
  - `DisplayAccountDetails()` (public): displays account details.

2. Create derived classes `SavingsAccount` and `CurrentAccount`:

- `SavingsAccount` should include an `InterestRate` attribute and a method to `CalculateInterest()`.
- `CurrentAccount` should include a method to `ChargeFee(double fee)`.

3. Implement a method to perform operations based on the type of the account using the `is`, `as`, and `typeof` operators:

- Use the `is` operator to check if an object is of a particular type before performing operations.
- Use the `as` operator to safely cast types and perform type-specific operations.
- Use the `typeof` operator to display the type of the account.

Output

- Display the account details, including the account type, and results of operations performed on the account.

Hints

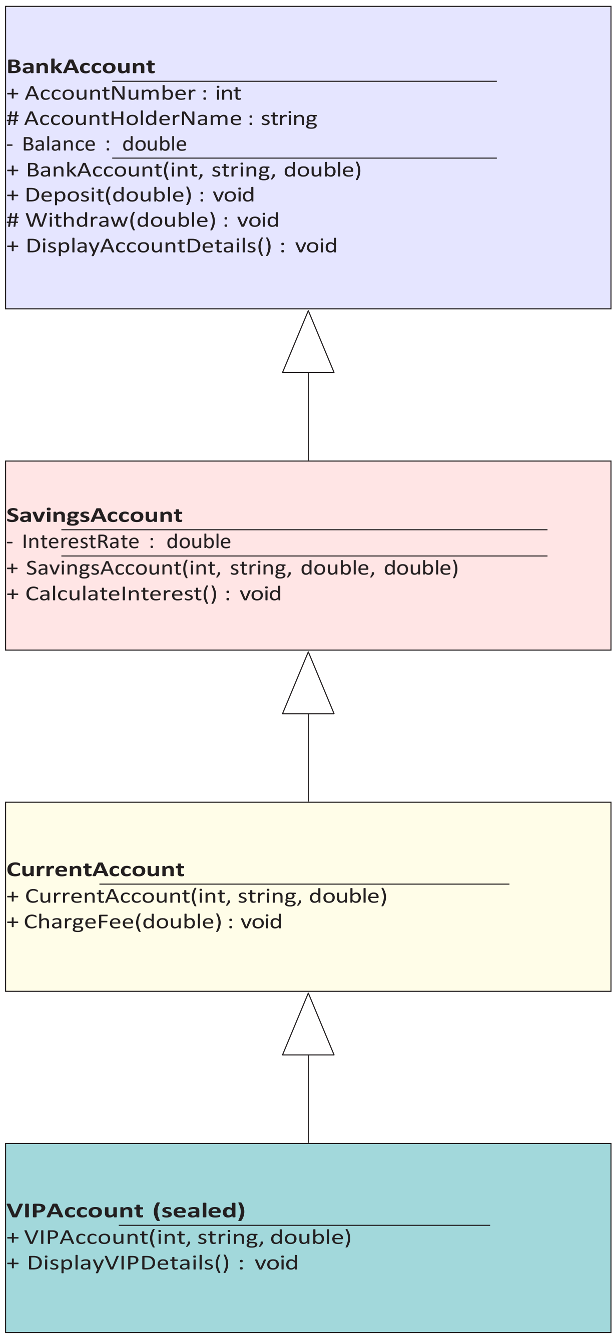
- Use the `is` operator to ensure that an object is of a specific type before performing type-specific operations.
- Use the `as` operator for safe casting, and check for `null` to avoid runtime exceptions.
- Use the `typeof` operator to get the type of an object at runtime.

Try

- Extend the application to include more account types (e.g., `FixedDepositAccount`) and implement additional type-specific operations.
- Implement a method to list all accounts and perform operations based on their types using type checking and casting.

7.8 Sealed Classes and Methods

Design and implement a bank application in C# to explore the use of sealed classes and methods to prevent further inheritance and method overriding. This exercise will help you understand how to restrict inheritance and control method overriding in object-oriented programming.


BankAccount
+ AccountNumber : int
# AccountHolderName : string
- Balance : double
+ BankAccount(int, string, double)
+ Deposit(double) : void
# Withdraw(double) : void
+ DisplayAccountDetails() : void
SavingsAccount
- InterestRate : double
+ SavingsAccount(int, string, double, double)
+ CalculateInterest() : void
CurrentAccount
+ CurrentAccount(int, string, double)
+ ChargeFee(double) : void
VIPAccount (sealed)
+ VIPAccount(int, string, double)
+ DisplayVIPDetails() : void

Input

- Bank account details such as account number, account holder's name, initial balance, account type, and specific attributes for derived account types (e.g., interest rate for savings accounts).
- Operations to perform on different account types, such as deposit, withdraw, calculate interest, and charge fees.

Process

1. Create a base class `BankAccount` with common attributes and methods:
    - `AccountNumber` (public): a unique identifier for the account.
    - `AccountHolderName` (protected): the name of the account holder.
    - `Balance` (private): the current balance of the account.

- Constructor to initialize `AccountNumber`, `AccountHolderName`, and `Balance`.
- Methods:
  - `Deposit(double amount)` (public): allows depositing money into the account.
  - `Withdraw(double amount)` (protected): allows withdrawing money from the account.
  - `DisplayAccountDetails()` (public): displays account details.

2. Create a derived class `SavingsAccount`:

   - `SavingsAccount` should include an `InterestRate` attribute and a method to `CalculateInterest()`.

3. Create a derived class `CurrentAccount`:

   - `CurrentAccount` should include a method to `ChargeFee(double fee)`.

4. Implement a `sealed` class `VIPAccount` that inherits from `BankAccount`:

   - `VIPAccount` should include additional privileges or features for VIP customers.
   - Ensure no other classes can inherit from `VIPAccount`.

5. Implement a `sealed` method in `BankAccount`:

   - Mark the `Withdraw` method as `sealed` to prevent overriding in derived classes.

Output

- Display the account details, including the account type, and results of operations performed on the account.

Hints

- Use the `sealed` keyword to prevent a class from being inherited.
- Use the `sealed` keyword in a method to prevent it from being overridden in derived classes.

Try

- Extend the application to include more account types (e.g., `FixedDepositAccount`) and implement additional type-specific operations without allowing inheritance from `VIPAccount`.
- Implement methods in the `VIPAccount` class that take advantage of its sealed nature.

Polymorphism, Abstract Classes and Interfaces

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

8.1 Complex Number Arithmetic

Demonstrate operator overloading by implementing addition for complex numbers. The program should define a `ComplexNumber` class, overload the `+` operator to perform addition of complex numbers, and display the result.

Program Requirements

1. Define a `ComplexNumber` class with properties for the real and imaginary parts.
2. Implement a constructor to initialize the real and imaginary parts of a complex number.
3. Overload the `+` operator to allow addition of two `ComplexNumber` objects.
4. Override the `ToString` method to provide a string representation of a complex number in the format "real + imaginaryi".
5. In the `Main` method, create instances of `ComplexNumber` and demonstrate the addition of two complex numbers using the overloaded `+` operator.
6. Print the result of the addition to the console.

Expected Output

The program should output the following:

```
Sum: 1.5 + 2.5i + 2.5 + 3.5i = 4 + 6i
```

8.2 Polynomial Arithmetic

Develop a C# program that demonstrates operator overloading by implementing multiplication for polynomials. The program should define a `Polynomial` class, overload the `*` operator to perform multiplication of polynomials, and display the result.

Requirements

1. Define a `Polynomial` class with a list to store the coefficients.
2. Implement a constructor to initialize the coefficients.
3. Overload the `*` operator to allow multiplication of two `Polynomial` objects.
4. Override the `ToString` method to provide a string representation of a polynomial in the standard mathematical format.
5. In the `Main` method, create instances of `Polynomial` and demonstrate the multiplication of two polynomials using the overloaded `*` operator.

6. Print the original polynomials and their product to the console.

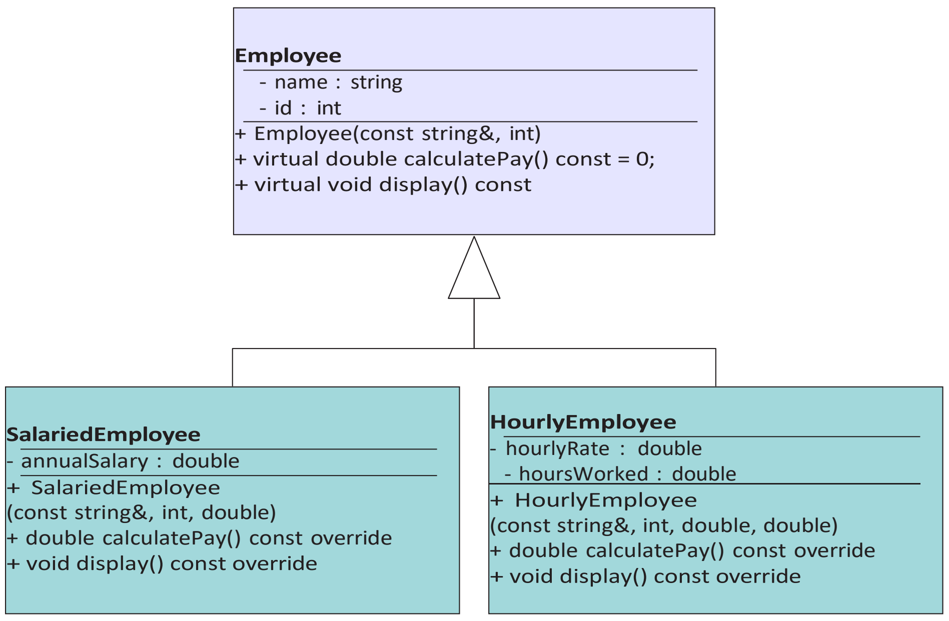
Expected Output

The program should output the following:

```
Polynomial 1:
3x^2 + 2x + 1
Polynomial 2:
5x^2 + 4x + 3
Product: 15x^4 + 22x^3 + 19x^2 + 10x + 3
```

8.3 Workforce Segments

The Workforce segment manages various types of employees and their specific payroll calculations. Implement a base class, `Employee`, and derive two classes, `SalariedEmployee` and `HourlyEmployee`, from it. This structure will allow efficient management of general employee information and specific payroll calculations for different types of employees.


Employee
- name : string
- id : int
+ Employee(const string&, int)
+ virtual double calculatePay() const = 0;
+ virtual void display() const
SalariedEmployee
- annualSalary : double
+ SalariedEmployee
(const string&, int, double)
+ double calculatePay() const override
+ void display() const override
HourlyEmployee
- hourlyRate : double
- hoursWorked : double
+ HourlyEmployee
(const string&, int, double, double)
+ double calculatePay() const override
+ void display() const override

Input

Employee details: Name, ID, and specific salary-related information.

Annual salary for salaried employees.

Hourly rate and hours worked for hourly employees.

Process Requirements

Define the `Employee` class with data members and member functions for general employee information.

Implement the `SalariedEmployee` class inheriting from `Employee`, with additional data members and member functions for salaried employees.

Implement the `HourlyEmployee` class inheriting from `Employee`, with additional data members and member functions for hourly employees.

Implement constructors, including default and parameterized, and appropriate member functions in each class.

Use polymorphism to calculate and display the pay for different types of employees.

Output

Display employee details and calculate their pay based on their type (salaried or hourly).

Hints

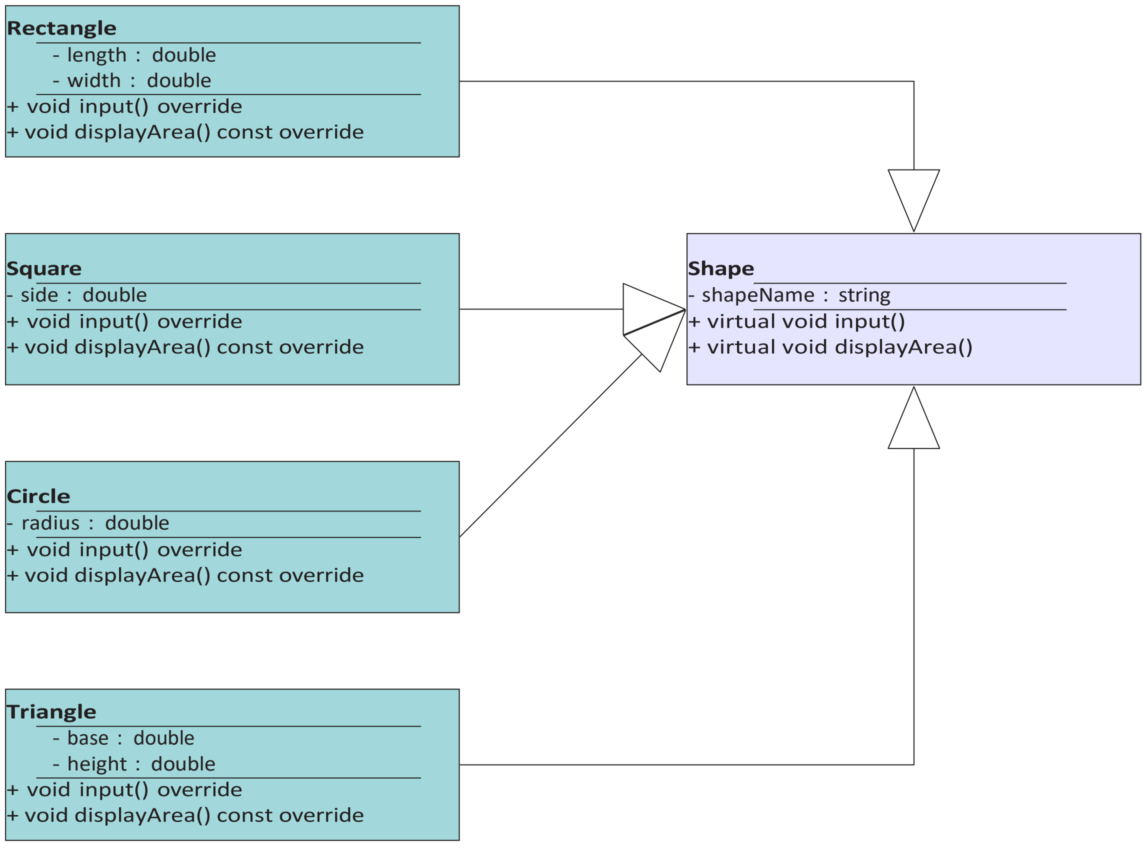
```
// Define base class Employee
// Define derived class SalariedEmployee
// Define derived class HourlyEmployee

// Usage example
SalariedEmployee se("Radha", 1001, 50000.0);
se.display();
cout << "Pay: " << se.calculatePay() << endl;

HourlyEmployee he("Smitha", 2002, 20.0, 40);
he.display();
cout << "Pay: " << he.calculatePay() << endl;
```

Try

Enhance the application by adding support for different types of employees (e.g., commission-based employees, contract employees) with specific payroll calculations.

8.4 Geometric Shapes: Abstract Class and Methods

Model different geometric shapes using object-oriented principles. Implement an abstract class `Shape` with the following specifications:



Shape Class:

- Define an abstract method `input()` to input dimensions specific to each shape.
- Define an abstract method `displayArea()` to calculate and display the area of the shape.
- The class should include a protected field `shapeName` to store the name of the shape.

Derived Classes:

Implement the following derived classes inheriting from `Shape`:

- `Rectangle`: Represents a rectangle with attributes for length and width.
- `Square`: Represents a square with a side length attribute.
- `Circle`: Represents a circle with a radius attribute.
- `Triangle`: Represents a triangle with attributes for base length and height.

Each derived class should override the `input()` method to accept user input for its specific dimensions and the `displayArea()` method to calculate and print its area.

Main Program:

In the `Main` method, demonstrate the usage of these shape classes:

- Create instances of each shape class.
- Prompt the user to input dimensions for each shape.
- Display the calculated area for each shape using the overridden `displayArea()`

method.

Expected Output

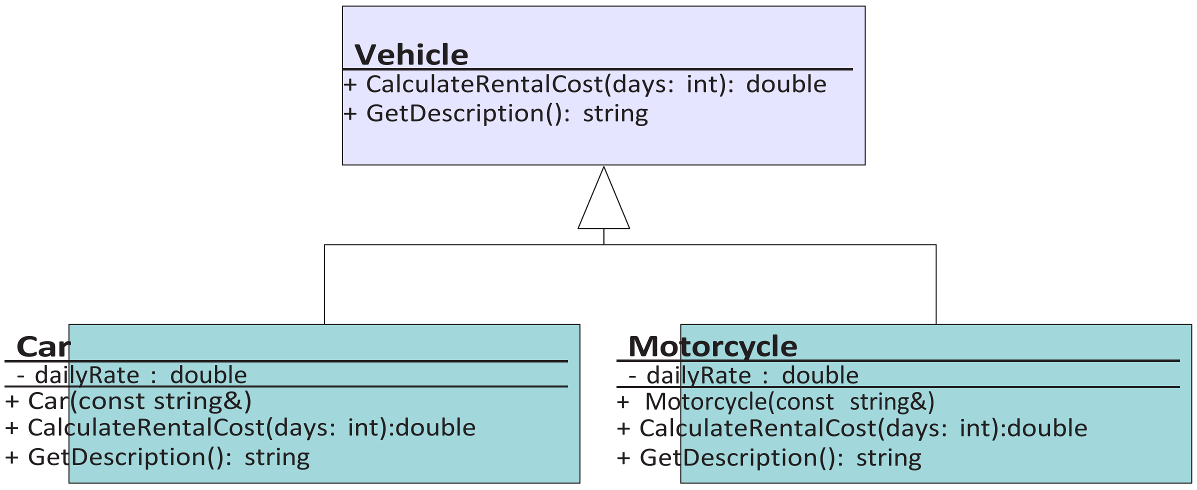
The program should output the calculated area for each shape based on user-provided dimensions.

Hints

- Use inheritance and method overriding to implement shape-specific behaviors.
- Ensure each derived class provides its own implementation of `input()` and `displayArea()`.
- Demonstrate polymorphism by treating all shapes uniformly through the base class reference.

8.5 Rental Vehicle Cost Calculator

You are developing a vehicle rental system where different types of vehicles can be rented for a specified number of days. Your task is to create a class hierarchy in C# to represent the vehicles and calculate the rental costs based on the type of vehicle and the number of rental days.


Vehicle
+ CalculateRentalCost(days: int): double
+ GetDescription(): string
Car
- dailyRate : double
+ Car(const string&)
+ CalculateRentalCost(days: int):double
+ GetDescription(): string
Motorcycle
- dailyRate : double
+ Motorcycle(const string&)
+ CalculateRentalCost(days: int):double
+ GetDescription(): string

Implementation Requirements

1. Define the abstract class `Vehicle` with the required abstract methods.
2. Define the `Car` class that extends `Vehicle` and implements the required methods.
3. Define the `Motorcycle` class that extends `Vehicle` and implements the required methods.
4. Ensure that the `Car` and `Motorcycle` classes correctly calculate the rental cost based on the number of days and their respective daily rates.
5. The `GetDescription()` method should return a string describing the vehicle, including its type and daily rate.

Example Usage

```
Vehicle car = new Car("Sedan", 50.0);
// Output: "Sedan with a daily rate of $50.00"
Console.WriteLine(car.GetDescription());
// Output: 250.0
Console.WriteLine(car.CalculateRentalCost(5));

Vehicle motorcycle = new Motorcycle("Sport Bike", 30.0);
// Output: "Sport Bike with a daily rate of $30.00"
Console.WriteLine(motorcycle.GetDescription());
// Output: 90.0
Console.WriteLine(motorcycle.CalculateRentalCost(3));
```

Hints

- Define an abstract class `Vehicle` with abstract methods `CalculateRentalCost` and `GetDescription`.

- Implement the `Car` and `Motorcycle` classes to inherit from `Vehicle`.

- Use the constructor in `Car` and `Motorcycle` to initialize the `dailyRate`.

- Ensure the `CalculateRentalCost` method in each derived class multiplies the `dailyRate` by the number of days.

- The `GetDescription` method should return a formatted string describing the vehicle.

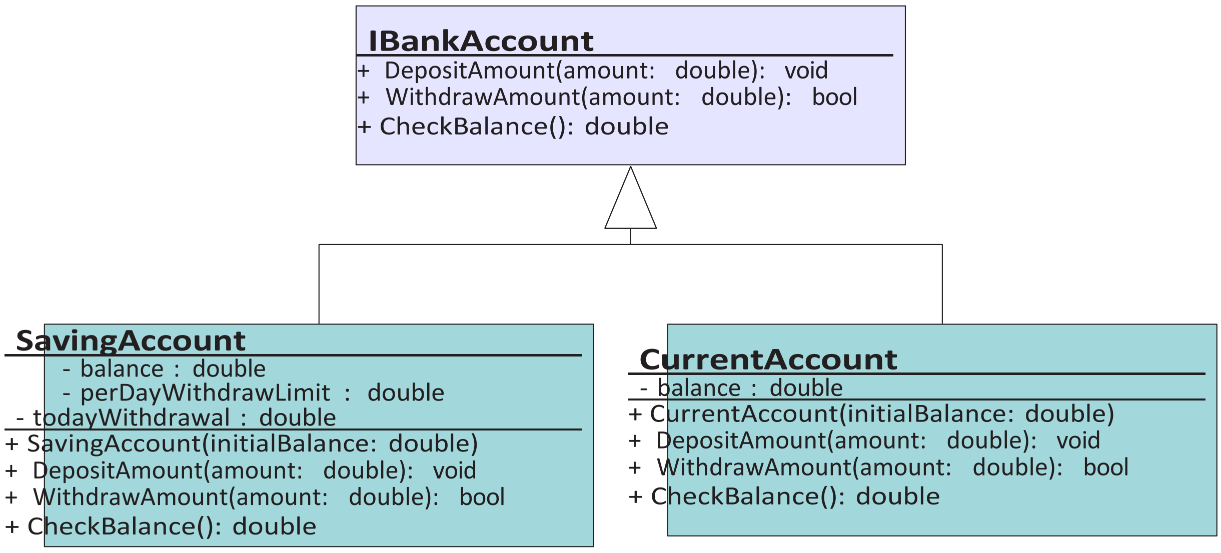
Note: Ensure that the implementation follows the principles of object-oriented program- ming, including encapsulation, inheritance, and polymorphism.

8.6 IBankAccount

Develop a bank application that supports the creation and management of the following account types:

Saving Account

Current Account


IBankAccount
+ DepositAmount(amount: double): void
+ WithdrawAmount(amount: double): bool
+ CheckBalance(): double
SavingAccount
- balance : double
- perDayWithdrawLimit : double
- todayWithdrawal : double
+ SavingAccount(initialBalance: double)
+ DepositAmount(amount: double): void
+ WithdrawAmount(amount: double): bool
+ CheckBalance(): double
CurrentAccount
- balance : double
+ CurrentAccount(initialBalance: double)
+ DepositAmount(amount: double): void
+ WithdrawAmount(amount: double): bool
+ CheckBalance(): double

Requirements:

The application should provide the following services for each account:

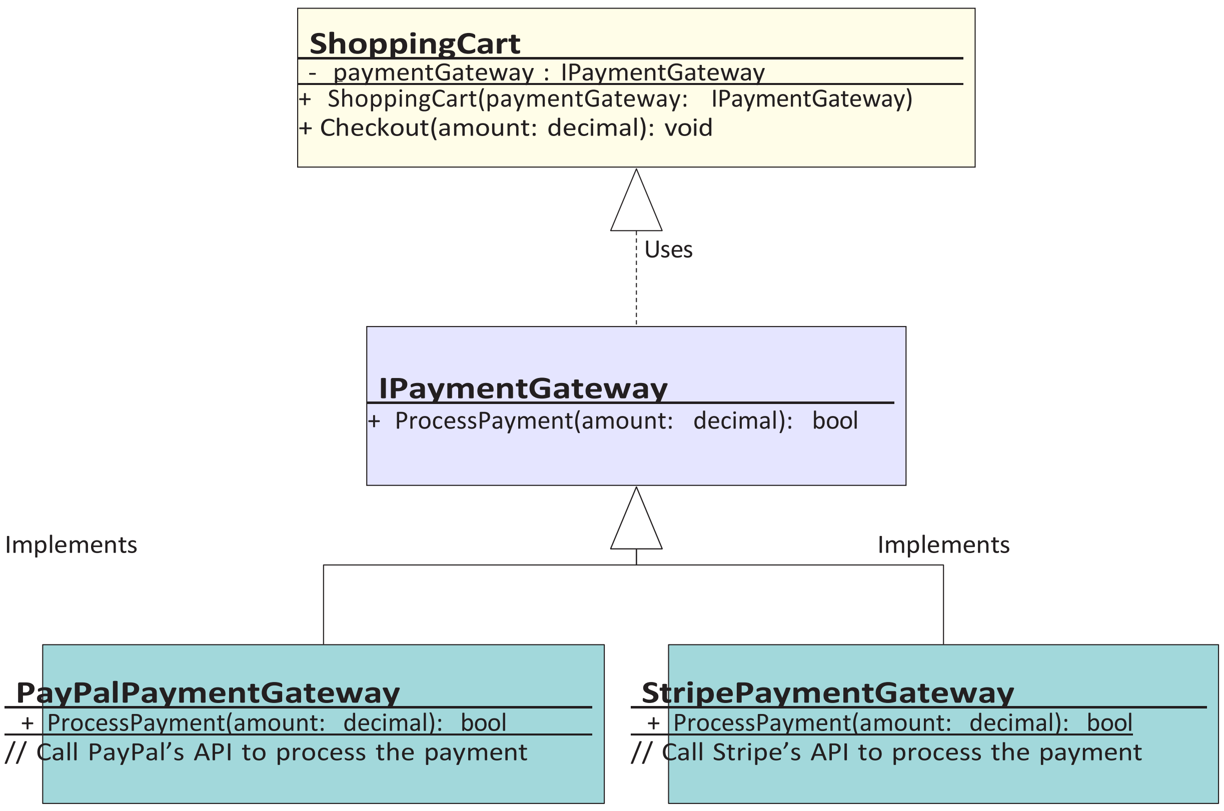
- **DepositAmount(amount: double): void** - Deposits the specified amount into the account.
- **WithdrawAmount(amount: double): bool** - Withdraws the specified amount from the account if sufficient funds are available.
- **CheckBalance(): double** - Returns the current balance of the account.

Hints

- Define the `IBankAccount` interface with `DepositAmount`, `WithdrawAmount`, and `CheckBalance` methods.
- Implement the `SavingAccount` class to handle balance, per-day withdrawal limit, and today's withdrawal.
- Implement the `CurrentAccount` class to handle balance without any withdrawal limit.
- Use the interface methods to perform deposit, withdrawal, and balance checking operations.
- Display the balance after each operation.
- Ensure that withdrawal operations check for sufficient funds before proceeding.

8.7 Payment Gateway

Develop a payment processing system using the `IPaymentGateway` interface. This interface should include a method `ProcessPayment` to handle payments. Implement this interface with two classes: `PayPalPaymentGateway` and `StripePaymentGateway`, each simulating pay- ment processing for their respective platforms.


ShoppingCart
- paymentGateway : IPaymentGateway
+ ShoppingCart(paymentGateway: IPaymentGateway)
+ Checkout(amount: decimal): void
Uses
IPaymentGateway
+ ProcessPayment(amount: decimal): bool
Implements
Implements
PayPalPaymentGateway
+ ProcessPayment(amount: decimal): bool
// Call PayPal's API to process the payment
StripePaymentGateway
+ ProcessPayment(amount: decimal): bool
// Call Stripe's API to process the payment

Create a `ShoppingCart` class that uses an `IPaymentGateway` to process payments. The system should be able to handle payments through different providers by swapping implementations.

Implement the `IPaymentGateway` interface and the associated payment provider classes. Ensure the `ShoppingCart` class can interact with any payment provider without modification. Test the implementation with both PayPal and Stripe providers.

Hints:

- The `ShoppingCart` class should not be concerned with the specifics of payment process- ing.
- Adding new payment providers should require minimal changes to the existing code.

Exception Handling

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

9.1 DivideByZeroException

Performs division of two numbers provided by the user. The program should handle exceptions gracefully, particularly when the user inputs non-numeric values.

Input:

Two inputs from the user representing the numbers to be divided:

- The first number (numerator).
- The second number (denominator).

Operations:

1. **Input Handling:**
   - Prompt the user to enter the first number (numerator).
   - Prompt the user to enter the second number (denominator).
2. **Exception Handling:**
   - Use a try-catch block to handle exceptions.
   - Catch `FormatException` to handle non-numeric inputs.
   - Catch `DivideByZeroException` to handle division by zero.
   - Optionally, catch other general exceptions to handle unexpected errors.
3. **Division Operation:**

- Perform the division operation if both inputs are valid numbers and the denominator is not zero.

4. **Output:**

   - If inputs are valid, display the result of the division.
   - If a `FormatException` occurs, display an error message indicating the input was not a valid number.
   - If a `DivideByZeroException` occurs, display an error message indicating that division by zero is not allowed.
   - Optionally, handle other exceptions and display a generic error message.

Hints:

- Use `Convert.ToDouble` or `double.TryParse` to convert the user input to a double type.
- Encapsulate the division logic within a try-catch block to handle potential exceptions.
- Consider edge cases like zero as a denominator and non-numeric inputs.

Sample Output:

- **Case 1: Valid Input**

```
Enter the first number: 10
Enter the second number: 2
Result: 10 / 2 = 5
```

- **Case 2: Non-Numeric Input**

```
Enter the first number: abc
Enter the second number: 2
Error: Input is not a valid number.
```

- **Case 3: Division by Zero**

```
Enter the first number: 10
Enter the second number: 0
Error: Division by zero is not allowed.
```

9.2 Handling Negative Number Exception

Define a method to check if a given integer is negative. If the integer is negative, the method should throw an exception. The calling code should handle this exception and provide appropriate feedback to the user.

Input:

- An integer value provided by the user.

Operations:

1. **Method Definition:**

   - Define a method `CheckIfNegative` that takes an integer as a parameter.
   - If the integer is negative, throw an exception with an appropriate message.

2. **Exception Handling:**

   - In the calling code, prompt the user to enter an integer.
   - Call the `CheckIfNegative` method with the user's input.
   - Use a `try-catch` block to handle the exception thrown by the method.
   - If an exception is caught, display an error message to the user.

Output:

- If the input integer is non-negative, display a message indicating that the number is valid.

- If the input integer is negative, catch the exception and display the error message.

Hints:

- Use the `throw` statement to raise an exception within the `CheckIfNegative` method.

- Use a `try-catch` block in the calling code to handle the exception and display an ap- propriate message to the user.

- Test the program with both positive and negative integers to ensure proper exception handling.

Sample Output:

- **Case 1: Non-Negative Input**

```
Enter an integer: 5
The number 5 is valid.
```

- **Case 2: Negative Input**

```
Enter an integer: -3
Error: The number -3 is negative.
```

9.3 Validating Integer Input Range

Prompt the user to input a numeric integer. The program should throw an exception if the number is less than 0 or greater than 1000. The calling code should handle the exception and display appropriate error messages.

Input:

- A numeric integer provided by the user.

Operations:

1. **Input Handling:**

   - Prompt the user to enter an integer.
   - Read the user input.

2. **Validation:**

   - Define a method `ValidateNumber` that takes an integer as a parameter.

- If the integer is less than 0 or greater than 1000, throw an exception with an appro- priate message.

3. **Exception Handling:**

   - Use a `try-catch` block to handle the exception thrown by the `ValidateNumber`

method.

   - If an exception is caught, display an error message to the user.

Output:

- If the input integer is between 0 and 1000 (inclusive), display a message indicating that the number is valid.
- If the input integer is less than 0 or greater than 1000, catch the exception and display the error message.

Hints:

- Use the `throw` statement to raise an exception within the `ValidateNumber` method.
- Use a `try-catch` block in the calling code to handle the exception and display an ap- propriate message to the user.
- Test the program with edge cases, such as -1, 0, 1000, and 1001, to ensure proper exception handling.

Sample Output:

- **Case 1: Valid Input**

```
Enter an integer: 500
The number 500 is valid.
```

- **Case 2: Less than 0**

```
Enter an integer: -5
Error: The number -5 is out of the allowed range (0-1000).
```

- **Case 3: Greater than 1000**

```
Enter an integer: 1500
Error: The number 1500 is out of the allowed range (0-1000).
```

9.4 Array Type Mismatch Exception

You are required to write a program that demonstrates the occurrence of an **ArrayTypeMis-matchException**. This exception is thrown when an array cannot store a given element because the actual type of the element is incompatible with the declared type of the array.

Input:

- Declare an array of integers.
- Attempt to assign a string value to an element in the integer array.

Process:

- Use a try-catch block to handle the exception.
- Catch the `ArrayTypeMismatchException` and display an error message indicating the attempted operation.

Output:

- Upon encountering the exception, output a descriptive error message explaining the nature of the exception.

Hints:

- Initialize an array of integers.
- Try assigning a string value or another incompatible data type to an element in the integer array.
- Encapsulate the assignment operation within a try block and handle the exception in the catch block.

9.5 Format Exception

Develop a program that illustrates the occurrence of a **FormatException**. This exception is thrown when an invalid format is detected in the input provided by the user or within the program.

Input:

- Prompt the user to enter a number as a string.
- Attempt to convert this string input to an integer using `int.Parse()` or `Convert.ToInt32()`.

Process:

- Use a try-catch block to handle the `FormatException`.
- Catch the exception and display an appropriate error message explaining the format error.

Output:

- Upon encountering the exception, output a clear error message indicating that the input string was not in a correct format.

Hints:

- Encourage users to provide valid numeric inputs.
- Use error handling techniques to gracefully manage incorrect format inputs.

9.6 Index Out of Range Exception

Design a program that demonstrates the occurrence of an **IndexOutOfRangeException**. This exception is thrown when an attempt is made to access an element of an array or a collection with an index that is outside the bounds of the array or collection.

Input:

- Declare an array with a specific size.
- Attempt to access an element using an index that is out of the array's bounds (e.g., accessing at an index greater than or equal to the array length).

Process:

- Use a try-catch block to handle the `IndexOutOfRangeException`.
- Catch the exception and display an informative error message explaining the index out of range error.

Output:

- Upon encountering the exception, output a descriptive error message indicating the at- tempted operation that caused the index out of range.

Hints:

- Initialize an array with a specific size and populate it with elements.
- Attempt to access an element using an index that exceeds the array's size.
- Utilize error handling techniques to gracefully manage index-related errors.

9.7 Invalid Cast Exception

Develop a program that demonstrates the occurrence of an **InvalidCastException**. This exception is thrown when an explicit conversion is attempted between incompatible data types, such as from a base type to a derived type or from one type to another that cannot be converted.

Input:

- Declare an object of a base type.
- Attempt to cast this object to a derived type that it is not compatible with.

Process:

- Use a try-catch block to handle the `InvalidCastException`.
- Catch the exception and display an informative error message explaining the invalid cast operation.

Output:

- Upon encountering the exception, output a descriptive error message indicating the at- tempted invalid cast.

Hints:

- Define classes or objects where an invalid cast scenario can be simulated.
- Attempt to cast an object to a type that it is not assignable to.
- Use error handling techniques to gracefully manage type casting errors.

9.8 Null Reference Exception

Design a program that demonstrates the occurrence of a **NullReferenceException**. This exception is thrown when an attempt is made to access or call a member (method or property) on an object reference that is currently null.

Input:

- Declare an object reference variable without initializing it (set it to null).
- Attempt to call a method or access a property on this null object reference.

Process:

- Use a try-catch block to handle the `NullReferenceException`.
- Catch the exception and display an informative error message explaining the null reference access.

Output:

- Upon encountering the exception, output a descriptive error message indicating the at- tempted null reference access.

Hints:

- Initialize an object reference variable to null.
- Attempt to call a method or access a property on this null reference without assigning an instance to it.
- Use error handling techniques to gracefully manage null reference errors.

9.9 Overflow Exception

Develop a program that demonstrates the occurrence of an **OverflowException**. This exception is thrown when an arithmetic operation exceeds the range of the data type used to store the result, resulting in an overflow.

Input:

- Declare variables of appropriate data types (e.g., `int`, `double`) and assign values that may cause overflow during arithmetic operations.

- Perform arithmetic operations that are likely to exceed the maximum or minimum value limits of the data type.

Process:

- Use a try-catch block to handle the `OverflowException`.
- Catch the exception and display an informative error message explaining the arithmetic overflow.

Output:

- Upon encountering the exception, output a descriptive error message indicating the arith- metic operation that caused the overflow.

Hints:

- Initialize variables with values that are near the maximum or minimum limits of their data type.
- Perform operations like addition, multiplication, or division that could exceed the data type's range.
- Use error handling techniques to gracefully manage arithmetic overflow errors.

9.10 Catching Multiple Exceptions

Develop a program that demonstrates how to catch multiple specific exceptions and use a general catch block for any other exceptions that are not explicitly handled.

Input:

- Prompt the user to enter two numbers for a division operation.
- Optionally, allow the user to choose an invalid operation to intentionally trigger specific exceptions.

Operations:

1. **Input Handling:**
   - Prompt the user to enter the first number (numerator).
   - Prompt the user to enter the second number (denominator).
2. **Exception Handling:**
   - Use a try-catch block to handle exceptions.
   - Catch `DivideByZeroException` to handle division by zero.
   - Catch `FormatException` to handle non-numeric inputs.
   - Catch `OverflowException` to handle arithmetic overflow.
   - Use a general catch block to handle any other unexpected exceptions.
3. **Division Operation:**
   - Perform the division operation if both inputs are valid numbers and the denominator is not zero.

Output:

- If inputs are valid, display the result of the division.
- If a `DivideByZeroException` occurs, display an error message indicating that division by zero is not allowed.
- If a `FormatException` occurs, display an error message indicating the input was not a valid number.
- If an `OverflowException` occurs, display an error message indicating an arithmetic overflow.
- If any other exception occurs, display a generic error message.

Hints:

- Use `Convert.ToInt32` or `int.TryParse` to convert the user input to an integer type.
- Encapsulate the division logic within a try-catch block to handle potential exceptions.
- Consider edge cases like zero as a denominator, non-numeric inputs, and very large numbers that may cause overflow.

File Handling

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

10.1 File Creation and Text Writing

You are required to develop a program that creates a new file named mytest.txt and writes some text content into this file. The program should ensure that the file is created successfully and the specified text is written into it.

Input:

No direct input from the user is required for this problem.

Operations:

1. Create a new file named `mytest.txt` in the specified directory.
2. Write the text "`Hello, this is a test file.`" into the file.
3. Ensure the file is saved properly with the written content.
4. Display a message indicating the successful creation of the file and the writing of the content.

Output:

The output should indicate the successful creation of the file and the addition of the text, as follows:

```
File mytest.txt created successfully with the following content: Hello
and Welcome to the C# Lab
In this lab we are working with Files. It’s really interesting to work
with C#
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use the `File.WriteAllText` method to create the file and write the specified content to it.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.2 Reading the content of the File

Demonstrate file reading operations. The program should read the content from the file and then display the content on the console. This will involve using file handling methods provided by the `System.IO` namespace to ensure proper reading of the file.

Input:

No direct input from the user is required for this problem.

Operations:

1. Search for the file named `mytest.txt` in the specified directory.
2. Read the content from the created file.
3. Display the content of the file on the console.

Output:

The output should display the content of the file, as follows:

```
Here is the content of the file mytest.txt:
Hello and Welcome to the C# Lab
In this lab we are working with
Files. It’s really interesting
to work with C#
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use the `File.WriteAllText` method to create the file and write the specified content to it.
- Use the `File.ReadAllText` method to read the content from the file.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.3 Writing an Array of Strings to a File

Create a text file and write an array of strings into it. The program should prompt the user to input the number of lines and the content for each line. It will then save this content to a file and display the file’s content.

Input:

- The number of lines to write to the file.
- The content for each line.

Operations/Instructions:

1. Prompt the user to input the number of lines to write in the file.
2. Prompt the user to input the content for each line.
3. Create a new text file named `myfile.txt`.
4. Write the user-provided lines into the file.
5. Read the content from the file.
6. Display the content of the file on the console.

Output:

The output should display the content of the file, as follows:

```
The content of the file is:
 --------------------------------------------
Explore new horizons every day.
Embrace challenges and seize
opportunities. Learn, grow, and
inspire others.
Dream big, work hard, achieve greatness.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use the `File.WriteAllLines` method to write the array of strings to the file.
- Use the `File.ReadAllLines` method to read the content from the file.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.4 Exclude lines that contain specific string

Create a text file and writes specified lines into it, excluding any line containing a specified string. The program should prompt the user to input the string to ignore, the number of lines to write, and the content for each line. It will then save this content to a file, excluding lines that contain the specified string, and display the remaining content.

Input:

- The string to ignore (lines containing this string will be excluded).
- The number of lines to write to the file.
- The content for each line.

Operations/Instructions:

1. Prompt the user to input the string to ignore in the lines.
2. Prompt the user to input the number of lines to write in the file.
3. Prompt the user to input the content for each line.
4. Create a new text file named `myfile.txt`.
5. Write the user-provided lines into the file, excluding any line that contains the specified string.
6. Read the content from the file.
7. Display the remaining content of the file on the console.

Output:

The output should display the remaining content of the file, as follows:

```
Input the string to ignore the line:
journey Input number of lines to write
in the file: 4
Input line 1: Happiness is a journey, not a
destination. Input line 2: Kindness costs
nothing but means everything.
Input line 3: Success is the sum of small efforts repeated day in and
day out. Input line 4: Courage is not the absence of fear; it's the
triumph over it.

The lines have been ignored which contain the string
'journey'. The content of the file is:
 ------------------------------------------
Kindness costs nothing but means everything.
Success is the sum of small efforts repeated day in and
day out. Courage is not the absence of fear; it's the
triumph over it.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use LINQ or simple loop with conditionals to filter out lines containing the specified string.
- Use the `File.WriteAllLines` method to write the lines to the file.
- Use the `File.ReadAllLines` method to read the content from the file.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.
-

10.5 Add some more

Demonstrate appending additional text to an existing file. Initially, the program will create a new text file and write some lines of text into it. Next, it will append more lines of text to the same file. The program should display the content of the file before and after appending.

Input:

Initial content to write to the file.

Additional content to append to the file.

Operations:

1. Create a new text file named `myfile.txt`.
2. Write the initial content provided by the user into the file.
3. Append the additional content provided by the user to the same file.

4. Read and display the content of the file before and after appending.

Output:

The output should display the content of the file before and after appending, as follows:

```
Initial content of the file:
 ------------------------------------------
This is the initial content of

the file. Content of the file

after appending:
 ------------------------------------------
This is the initial content of the file. Now appending some more
text to the file.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use `File.WriteAllText` to write the initial content to the file.
- Use `File.AppendAllText` to append additional content to the file.
- Use `File.ReadAllText` to read the content from the file.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.6 Snapshot

Create a snapshot by copying the entire content of one text file to another. The program should prompt the user to specify the source file and the destination file. It will then read the content from the source file and write it into the destination file, preserving the original content. Finally, the program should display the content of the copied file.

Input:

- Source file path (from which content will be copied).
- Destination file path (where content will be copied).

Operations:

1. Prompt the user to input the path of the source file.
2. Prompt the user to input the path of the destination file.
3. Read the entire content from the source file.
4. Write the read content into the destination file.
5. Read the content from the destination file.
6. Display the content of the copied file on the console.

Output:

The program should output the content of the copied file on the console after copying the content from the source file to the destination file.

```
Input the source file path: C:\myfolder\sourcefile.txt
Input the destination file path: C:\myfolder\destinationfile.txt

Successfully copied contents from ’sourcefile.txt’ to

’destinationfile.txt’.

Content of ’destinationfile.txt’:
 -------------------------------------------
This is the content of the source file that has been copied.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use `File.ReadAllText` to read the content from the source file.
- Use `File.WriteAllText` to write the content to the destination file.
- Use `File.ReadAllText` again to read the content from the destination file for display.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.7 Count, Top & Tail

Count the number of lines in a text file and displays both the first (top) and last (tail) lines of the file. The program should prompt the user to input the file path. It will then read the content from the file, count the lines, and output the first and last lines along with the total number of lines in the file.

Input:

- File path of the text file.

Operations/Instructions:

1. Prompt the user to input the path of the text file.
2. Read the entire content of the file.
3. Count the number of lines in the file.
4. Display the first line of the file.
5. Display the last line of the file.
6. Display the total number of lines in the file.

Output:

```
Input the file path: C:\myfolder\sample.txt

Content of the file is:
 --------------------------------------------------------------------
   1. Positivity means focusing on the good in every situation.
   2. It involves staying optimistic and hopeful.
   3. Positivity helps in overcoming challenges with resilience.
   4. It promotes a sense of inner peace and calmness.
   5. Cultivating positivity leads to a happier and more fulfilling
      life.


First line of the file:
 ----------------------------------------------
   1. Positivity means focusing on the good in every situation.

Last line of the file:
 ----------------------------------------------
   5. Cultivating positivity leads to a happier and more fulfilling
      life.

Total number of lines in the file: 5
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use `File.ReadAllLines` to read all lines from the file into an array.
- Access the first element (`lines[0]`) and last element (`lines[lines.Length - 1]`) of the array to get the first and last lines.
- Use `lines.Length` to get the total number of lines.
- Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.8 Selective File Reading

Read a specific line or set of lines from a text file based on user input. The program should prompt the user to input the file path and the line number(s) they wish to retrieve. It will then open the file, read the specified line(s), and display them to the user.

Input:

- File path of the text file.
- Line number(s) to read from the file (e.g., single line, multiple lines).

Operations:

1. Prompt the user to input the path of the text file.
2. Prompt the user to input the line number(s) they want to read.

3. Read the specified line(s) from the file.

4. Display the content of the selected line(s) to the user.

Output:

The program should output the content of the selected line(s) from the file based on the user's input.

```
Input the file path: C:\myfolder\sample.txt

Input the line number(s) to read (e.g., 1,

3, 5): 2 Content of line 2 in 'sample.txt':
 ---------------------------------------------
   2. It involves staying optimistic and hopeful.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use `StreamReader` to read the file and `ReadLine()` method to navigate to specific lines.
- Handle exceptions such as file not found or invalid line numbers using `try-catch` blocks.

10.9 n-Tail Extract

Read and display the last *n* lines from a text file. The program should prompt the user to input the file path and the number of lines they wish to retrieve from the end of the file. It will then read the file, extract the specified number of lines from the end, and display them.

Input:

File path of the text file.

Number of lines *n* to read from the end of the file.

Operations:

1. Prompt the user to input the path of the text file.
2. Prompt the user to input the number of lines *n* they want to read from the end of the file.
3. Read the entire content of the file.
4. Extract the last *n* lines from the file content.
5. Display the extracted lines to the user.

Output:

The program should output the last *n* lines from the specified text file.

```
Input the file path: C:\myfolder\sample.txt
Input the number of lines to read from the end of the file: 3

Content of the last 3 lines in 'sample.txt':
 --------------------------------------------
   3. Positivity helps in overcoming challenges with resilience.
   4. It promotes a sense of inner peace and calmness.
   5. Cultivating positivity leads to a happier and more fulfilling
      life.
```

Hints:

Use the `System.IO` namespace to handle file operations.

Use `File.ReadAllLines` to read all lines from the file into an array.

Use array slicing to access the last *n* elements of the array.

Handle any potential exceptions using a `try-catch` block to ensure the program does not crash in case of an error.

10.10 Strip Words

Remove specific words from a text file based on a list of stop words provided in another file. The program should prompt the user to input the file paths of both the text file and the stop words file. It will then read both files, remove all instances of the stop words from the text content, and save the cleaned content to a new file.

Input:

- File path of the input text file.
- File path of the stop words file.
- File path to save the cleaned text file.

Operations:

1. Prompt the user to input the path of the input text file.
2. Prompt the user to input the path of the stop words file.
3. Prompt the user to input the path to save the cleaned text file.
4. Read the content of the input text file.
5. Read the list of stop words from the stop words file.
6. Remove all instances of the stop words from the text content.
7. Write the cleaned text to the specified output file.
8. Display a message indicating that the process is complete and the

cleaned text has been saved.

Output:

The program should output the cleaned text to the specified output file, excluding all words listed in the stop words file.

```
Input the file path of the input text file:
C:\myfolder\input.txt Input the file path of the stop words
file: C:\myfolder\stopwords.txt
Input the file path to save the cleaned text file:
C:\myfolder\output.txt

Stop words removed
successfully. Cleaned
content saved to
'output.txt'.
```

Hints:

- Use the `System.IO` namespace to handle file operations.
- Use `File.ReadAllText` to read the file content and `File.ReadAllLines` to read the stop words.
- Use string manipulation methods to remove the stop words from the text.
- Handle exceptions using a `try-catch` block to manage potential errors, such as file not found or access issues.

Note:

Verify the files**Foundations of LINQ and Generic Types**

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

11.1 Filtering with LINQ

Showcase the use of LINQ to filter even numbers from a given list of integers.

Input:

You are provided with a list of integers.

Operations:

1. Implement a LINQ query to filter out only the even numbers from the list.

2. Display or output the filtered list of even numbers.

Output:

For example, given the input list `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`, the program should output:

```
Even numbers: 2, 4, 6, 8, 10
```

Hints:

- Use LINQ's `Where` method along with a lambda expression to filter numbers.
- Ensure to handle edge cases such as an empty list or no even numbers present in the list.

11.2 Sorting Objects with LINQ

Implement a LINQ query to sort a list of objects by a property in ascending order.

Input:

You are provided with a list of objects.

Operations:

1. Implement a LINQ query to sort the list of objects by a specified property in ascending order.
2. Display or output the sorted list of objects.

Output:

For example, given a list of objects with a property `Age`, the program should output:

```
Sorted list by Age: [Object1, Object2, Object3, ...]
```

Hints:

- Use LINQ's `OrderBy` method to sort objects by the specified property.
- Handle cases where the list may be empty or the property values are null.

11.3 Grouping and Counting with LINQ

Group a list of students by their age and count the number of students in each group using LINQ's `GroupBy` and `Count` methods.

Input:

You are provided with a list of student objects, each containing an `Age` property.

Operations:

1. Implement a LINQ query to group students by their `Age` property.

2. Calculate and display the count of students in each age group.

Output:

For example, given a list of students:

```
[
{ Name: "Surya", Age: 21 },
{ Name: "Keerthi", Age: 22 },
{ Name: "Venkatesh", Age: 21 },
{ Name: "Anusha", Age: 22 },
{ Name: "Padma", Age: 23 }
]
```

The program should output:

```
Age 21: 2 students
Age 22: 2 students
Age 23: 1 student
```

Hints:

- Use LINQ's `GroupBy` method to group objects by a property.
- Use LINQ's `Count` method within the grouped result to determine the number of items in each group.
- Handle cases where the list may be empty or where age values may be null.

11.4 Aggregation with LINQ

Calculate the sum and average of a list of numeric values using LINQ's aggregate functions.

Input:

You are provided with a list of numeric values.

Operations:

1. Implement a LINQ query to calculate the sum of the numeric values.
2. Implement a LINQ query to calculate the average of the numeric values.

Output:

For example, given a list of numeric values:

```
[10, 20, 30, 40, 50]
```

The program should output:

```
Sum: 150
Average: 30
```

Hints:

- Use LINQ's `Sum` method to compute the sum of elements in the list.

- Use LINQ's `Average` method to compute the average of elements in the list.
- Ensure to handle cases where the list may be empty or where values are null.

11.5 Distinct Values with LINQ

Remove duplicate strings from an array using LINQ's `Distinct` method.

Input:

You are provided with an array of strings.

Operations:

1. Implement a LINQ query to remove duplicate strings from the array.

Output:

For example, given an array of strings:

```
["apple", "orange", "banana", "apple", "grapes", "banana"]
```

The program should output:

```
Distinct strings: ["apple", "orange", "banana", "grapes"]
```

Hints:

- Use LINQ's `Distinct` method to filter out duplicate elements from the array.
- Maintain the order of elements in the resulting sequence.
- Handle cases where the array may be empty or contain null values.

11.6 Pagination with LINQ

Implement paging functionality to retrieve a specific number of records from a collection using LINQ's `Skip` and `Take` methods.

Input:

You are provided with a collection of records (e.g., a list or array).

Operations:

1. Implement a LINQ query to skip a specified number of records (offset) and take a specified number of records (page size).

Output:

For example, given a collection of records:

```
["Record1", "Record2", "Record3", "Record4", "Record5", "Record6"]
```

If you want to display records for page 2 with page size 2, the program should output:

```
["Record3", "Record4"]
```

Hints:

- Use LINQ's `Skip` method to skip a specified number of elements.
- Use LINQ's `Take` method to retrieve a specified number of elements after skipping.
- Ensure to handle cases where the collection may be empty or the specified page and page size exceed the collection length.

11.7 Conditional Filtering with LINQ

Filter out strings containing a specific substring from a list of strings using LINQ.

Input:

You are provided with a list of strings.

Operations:

1. Implement a LINQ query to filter strings that contain a specified substring.

Output:

For example, given a list of strings:

```
["apple", "orange", "banana", "pineapple", "grapes", "kiwi"]
```

If the substring to filter is "apple", the program should output:

```
Filtered strings: ["apple", "pineapple"]
```

Hints:

- Use LINQ's `Where` method with a lambda expression to filter strings based on the sub- string condition.
- Ensure to handle cases where the list may be empty or where the substring might not match any strings in the list.

11.8 Projecting Data with LINQ

Select only the names of students from a list of student objects using LINQ's `Select` clause.

Input:

You are provided with a list of student objects, each containing a `Name` property.

Operations:

1. Implement a LINQ query to extract and project only the names of students from the list.

Output:

For example, given a list of student objects:

```
[
{ Name: "Surya", Age: 21 },
{ Name: "Keerthi", Age: 22 },
{ Name: "Venkatesh", Age: 21 },
{ Name: "Anusha", Age: 22 },
{ Name: "Padma", Age: 23 }
]
```

The program should output:

```
Student names: ["Surya", "Keerthi", "Venkatesh", "Anusha", "Padma"]
```

Hints:

- Use LINQ's `Select` method to project the `Name` property from each student object.
- Ensure to handle cases where the list may be empty or where student names might be null or empty strings.

11.9 Join Operations with LINQ

Perform an inner join between two lists of objects based on a common property using LINQ.

Input:

You are provided with two lists of objects, each containing a common property for joining (e.g., Student objects with a common property like `Age`).

Operations:

1. Implement a LINQ query to perform an inner join between the two lists based on the common property.

Output:

For example, given two lists of objects:

```
List A:
[
{ Name: "Surya", Age: 21 },
{ Name: "Keerthi", Age: 22 },
{ Name: "Venkatesh", Age: 21 },
{ Name: "Anusha", Age: 22 },
{ Name: "Padma", Age: 23 }
]

List B:
[
{ Name: "Ravi", Age: 21 },
{ Name: "Kiran", Age: 22 },
{ Name: "Vijaya", Age: 23 },
{ Name: "Ramya", Age: 24 }
]
```

If joining based on the `Age` property, the program should output:

```
Inner join result:
[
{ NameA: "Surya", NameB: "Ravi", Age: 21 },
{ NameA: "Keerthi", NameB: "Kiran", Age: 22 },
{ NameA: "Padma", NameB: "Vijaya", Age: 23 }
]
```

Hints:

- Use LINQ's `Join` method along with a lambda expression to perform the inner join.
- Ensure to handle cases where there are no common elements between the lists or where the lists may be empty.

11.10 Handling Null Values with LINQ

Safely handle null values when selecting data using LINQ's null-conditional operators (`?.` and `??`).

Input:

You are provided with a collection of objects, some of which may contain null values for certain properties.

Operations:

1. Implement a LINQ query to select data from the collection, ensuring safe navigation through potentially null properties using `?.`.
2. Use the null-coalescing operator `??` to provide default values or handle null cases gracefully.

Output:

For example, given a collection of objects:

```
[
{ Name: "Ravi", Age: 30 },
{ Name: "Priya", Age: null },
{ Name: "Arun", Age: 25 },
{ Name: null, Age: 28 }
]
```

The program should output:

```
Selected data with handling null values:
[
{ Name: "Ravi", Age: 30 },
{ Name: "Priya", Age: 0 },         % Assume default value 0 for Age
{ Name: "Arun", Age: 25 },
{ Name: "", Age: 28 }              % Assume empty string "" for Name
]
```

Hints:

- Use LINQ's null-conditional operator `?.` to safely navigate through potentially null prop- erties.
- Use the null-coalescing operator `??` to provide default values or handle null cases.
- Handle scenarios where entire objects might be null in the collection.

Advanced Methods and Collections

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

12.1 Advanced Sorting with LINQ

Sort a list of objects by multiple criteria using LINQ's `OrderBy` and `ThenBy` methods.

Input:

You are provided with a list of objects, each containing multiple properties for sorting (e.g., Student objects with properties like `Name`, `Age`, and `Grade`).

Operations:

1. Implement a LINQ query to sort the list of objects primarily by one property using
`OrderBy` and secondarily by another property using `ThenBy`.

Output:

For example, given a list of student objects:

```
[
{ Name: "Surya", Age: 21, Grade: "A" },
{ Name: "Keerthi", Age: 22, Grade: "B" },
{ Name: "Venkatesh", Age: 21, Grade: "C" },
{ Name: "Anusha", Age: 22, Grade: "A" },
{ Name: "Padma", Age: 23, Grade: "B" }
]
```

The program should output:

```
Sorted list:
[
{ Name: "Anusha", Age: 22, Grade: "A" },
{ Name: "Keerthi", Age: 22, Grade: "B" },
{ Name: "Padma", Age: 23, Grade: "B" },
{ Name: "Surya", Age: 21, Grade: "A" },
{ Name: "Venkatesh", Age: 21, Grade: "C" }
]
```

Hints:

- Use LINQ's `OrderBy` method to sort by the primary property and `ThenBy` method to sort by the secondary property.

- Handle cases where the list may be empty or where sorting criteria may include null values or unexpected data types.

12.2 Grouping and Aggregation with LINQ

Group a list of sales transactions by month and calculate the total sales amount for each month using LINQ.

Input:

You are provided with a list of sales transactions, each containing a `Date` (representing the transaction date) and `Amount` (representing the sales amount).

Operations:

1. Implement a LINQ query to group the sales transactions by month based on the `Date`
property.
2. Calculate the total sales amount for each month using LINQ's aggregation functions.

Output:

For example, given a list of sales transactions:

```
[
{ Date: "2024-01-05", Amount: 100.00 },
{ Date: "2024-02-15", Amount: 150.00 },
{ Date: "2024-01-20", Amount: 120.00 },
{ Date: "2024-03-10", Amount: 200.00 },
{ Date: "2024-02-28", Amount: 180.00 }
]
```

The program should output:

```
Sales totals by month:
January: 220.00
February: 330.00
March: 200.00
```

Hints:

- Use LINQ's `GroupBy` method to group transactions by month.
- Use LINQ's aggregation methods like `Sum` to calculate the total sales amount for each group.
- Handle cases where there are no transactions for a particular month or where the list may be empty.

12.3 Joining Complex Data with LINQ

Perform an outer join between two lists of objects with a fallback value for missing matches using LINQ’s `Join` and `DefaultIfEmpty` methods.

Input:

You are provided with two lists of objects, each containing properties that can be matched for joining (e.g., Employee objects with a common property like `DepartmentId`).

Operations:

1. Implement a LINQ query to perform an outer join between the two lists based on the common property.
2. Use `DefaultIfEmpty` to provide a fallback value when no matches are found in the second list.

Output:

For example, given two lists of objects:

```
List A:
[
{ EmployeeId: 1, Name: "Ravi", DepartmentId: 101 },
{ EmployeeId: 2, Name: "Priya", DepartmentId: 102 },
{ EmployeeId: 3, Name: "Arun", DepartmentId: 103 }
]

List B:
[
{ DepartmentId: 101, DepartmentName: "IT" },
{ DepartmentId: 103, DepartmentName: "HR" }
]
```

If joining based on the `DepartmentId` property, the program should output:

```
Outer join result:
[
{ EmployeeId: 1, Name: "Ravi", DepartmentId: 101, DepartmentName:
"IT" },
{ EmployeeId: 2, Name: "Priya", DepartmentId: 102, DepartmentName:
null },
{ EmployeeId: 3, Name: "Arun", DepartmentId: 103, DepartmentName:
"HR" }
]
```

Hints:

- Use LINQ’s `Join` method to perform the outer join and `DefaultIfEmpty` method to handle missing matches.
- Ensure to handle cases where there are no common elements between the lists or where the lists may be empty.

12.4 Union and Concatenation with LINQ

Combine two lists into one unique list using LINQ's `Union` and `Concat` methods.

Input:

You are provided with two lists of objects.

Operations:

1. Implement a LINQ query to combine the two lists into one unique list.
2. Use `Union` to ensure that duplicate elements are removed and `Concat` to concatenate both lists while preserving duplicates.

Output:

For example, given two lists:

```
List A: [1, 2, 3, 4]
List B: [3, 4, 5, 6]
```

The program should output:

```
Unique combined list (Union):
[1, 2, 3, 4, 5, 6]

Concatenated list (Concat):
[1, 2, 3, 4, 3, 4, 5, 6]
```

Hints:

- Use LINQ's `Union` method to merge the lists and remove duplicates.
- Use LINQ's `Concat` method to concatenate both lists while preserving duplicates.
- Handle cases where the lists may be empty or contain null elements.

12.5 Element Manipulation with LINQ

Use LINQ to reverse the order of elements in a list and select specific elements based on index.

Input:

You are provided with a list of elements (e.g., integers, strings).

Operations:

1. Implement a LINQ query to reverse the order of elements in the list.
2. Select specific elements from the reversed list based on given indices (e.g., select elements at indices 0, 2, 4).

Output:

For example, given a list of integers:

```
Input list: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Indices to select: [0, 2, 4]
```

The program should output:

```
Reversed list: [9, 8, 7, 6, 5, 4, 3, 2, 1]
Selected elements: [9, 7, 5]
```

Hints:

- Use LINQ's `Reverse` method to reverse the order of elements in the list.
- Use LINQ's `Where` method with a condition based on index to select specific elements.
- Handle cases where the list may be empty or where the indices provided may exceed the list length.

12.6 Subqueries with LINQ

Write a LINQ query that includes a subquery to retrieve data from related tables or nested collections.

Input:

You are provided with two related collections or tables, where one collection/table contains objects that reference objects in the other collection/table.

Operations:

1. Implement a LINQ query that includes a subquery to retrieve data from the related tables or nested collections.
2. Use appropriate LINQ operators to perform operations such as filtering, projecting, or aggregating data from the subquery.

Output:

For example, given two collections of objects:

```
List A:
[
{ EmployeeId: 1, Name: "Ravi", DepartmentId: 101 },
{ EmployeeId: 2, Name: "Priya", DepartmentId: 102 },
{ EmployeeId: 3, Name: "Arun", DepartmentId: 103 }
]

List B:
[
{ DepartmentId: 101, DepartmentName: "IT" },
{ DepartmentId: 102, DepartmentName: "Finance" },
{ DepartmentId: 103, DepartmentName: "HR" }
]
```

Implement a LINQ query to retrieve employees along with their department names:

```
LINQ query result:
[
{ EmployeeId: 1, Name: "Ravi", DepartmentId: 101, DepartmentName:
"IT" },
{ EmployeeId: 2, Name: "Priya", DepartmentId: 102, DepartmentName:
"Finance" },
{ EmployeeId: 3, Name: "Arun", DepartmentId: 103, DepartmentName:
"HR" }
]
```

Hints:

- Use LINQ's `Join` or `SelectMany` methods to perform the subquery operation.
- Ensure to handle cases where there are no common elements between the collections or where the collections may be empty.

12.7 Error Handling in LINQ Queries

Handle exceptions and errors gracefully in LINQ queries, particularly when dealing with null values or unexpected data.

Input:

You are provided with data that may include null values or unexpected formats.

Operations:

1. Implement a LINQ query that gracefully handles potential exceptions or errors that may arise during data processing.
2. Use error-handling techniques such as null checks, exception handling blocks, or default value assignments to manage unexpected scenarios.

Output:

For example, given a list of integers with potential null values:

```
Input list: [1, null, 3, 4, null, 6, 7, 8, null]
```

The program should handle null values and unexpected data formats gracefully, ensuring the query operates without throwing exceptions.

Hints:

- Use LINQ's `Where` method with null checks or `Select` method with conditional operators to filter or transform data while handling nulls.
- Consider using try-catch blocks around LINQ queries to catch and manage exceptions that may occur during data processing.

12.8 Custom Filtering using LINQ and Lambda Expressions

Implement custom filtering of a generic collection of objects dynamically based on user-defined criteria using LINQ and lambda expressions.

Input:

You are provided with a generic collection of objects.

Operations:

1. Implement a LINQ query that allows dynamic filtering of objects based on user-defined criteria.
2. Use lambda expressions to construct flexible filtering conditions, such as filtering by specific properties or applying complex logical operations.

Output:

For example, given a list of `Product` objects:

```
List of Products:
[
{ Id: 1, Name: "Laptop", Price: 1200 },
{ Id: 2, Name: "Smartphone", Price: 800 },
{ Id: 3, Name: "Tablet", Price: 500 }
]
```

Implement a LINQ query that allows filtering products based on user-defined criteria, such as products with a price greater than 600:

```
Filtered list of Products (Price > 600):
[
{ Id: 1, Name: "Laptop", Price: 1200 },
{ Id: 2, Name: "Smartphone", Price: 800 }
]
```

Hints:

- Use LINQ's `Where` method with lambda expressions to apply dynamic filtering conditions.
- Allow users to input filtering criteria interactively or define them programmatically.

12.9 Dynamic Queries with LINQ

Build dynamic LINQ queries based on runtime conditions and user inputs using expression trees and Queryable extensions.

Input:

You are provided with a dataset that needs to be queried dynamically based on user-defined conditions.

Operations:

1. Implement a LINQ query that can dynamically adjust its filtering, sorting, or projection based on runtime conditions.
2. Utilize expression trees and Queryable extensions to construct LINQ queries programmat- ically.

Output:

For example, given a dataset of `Employee` objects:

```
List of Employees:
[
{ Id: 1, Name: "Ravi", Department: "HR", Salary: 50000 },
{ Id: 2, Name: "Priya", Department: "IT", Salary: 60000 },
{ Id: 3, Name: "Arun", Department: "Finance", Salary: 55000 }
]
```

Implement a LINQ query that can dynamically filter employees based on user inputs (e.g., filter by department and salary range):

```
Filtered list of Employees (Department: IT, Salary > 55000): [
{ Id: 2, Name: "Priya", Department: "IT", Salary: 60000 }
]
```

Hints:

- Use expression trees to build LINQ queries dynamically based on conditions such as user input.
- Explore LINQ's Queryable extensions to handle sorting, paging, and complex filtering scenarios dynamically.

12.10 Parallel LINQ (PLINQ)

Write LINQ queries that leverage PLINQ for parallel execution and improved performance.

Input:

You are provided with a dataset that can benefit from parallel processing.

Operations:

1. Implement LINQ queries using PLINQ to parallelize operations such as filtering, sorting, or aggregation.
2. Utilize PLINQ's parallel execution capabilities to enhance performance on large datasets.

Output:

For example, given a dataset of `Product` objects:

```
List of Products:
[
{ Id: 1, Name: "Laptop", Price: 1200 },
{ Id: 2, Name: "Smartphone", Price: 800 },
{ Id: 3, Name: "Tablet", Price: 500 }
]
```

Implement a PLINQ query that parallelizes operations like filtering or aggregation:

```
Filtered list of Products (Price > 600
using PLINQ): [
{ Id: 1, Name: "Laptop", Price: 1200 },
{ Id: 2, Name: "Smartphone", Price: 800 }
]
```

Hints:

- Use PLINQ's `AsParallel` method to enable parallel execution of LINQ queries.
- Consider the benefits and limitations of parallel processing, such as thread safety and overhead.

Explore Threads

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

13.1 Thread Life Cycle

Develop a C# program that demonstrates the lifecycle management of a worker thread, including starting, suspending, resuming, and terminating the thread.

Requirements

1. Create a new thread and start its execution.
2. Pause the main thread for a specific duration to simulate a wait period.
3. Suspend the worker thread after a designated time interval.
4. Resume the suspended thread after another wait period.
5. Terminate the worker thread gracefully using appropriate thread abort mechanisms.
6. Display messages to indicate each phase of the thread's lifecycle.

Expected Output

The program should output messages indicating the various stages of the thread's lifecycle, including starting, suspending, resuming, and terminating:

```
Main thread: Thread
started. Worker
thread: Starting
work... Worker
thread: Working...
Part 1 Worker thread:
Working... Part 2
Main thread: Suspending the worker
thread... Main thread: Resuming the
worker thread...
Worker thread: Working...
Part 3 Worker thread:
Working... Part 4
Main thread: Aborting the worker
thread... Worker thread:
Working... Part 5
Exception thrown: 'System.Threading.ThreadAbortException' in
mscorlib.dll Worker thread: Aborted.
Exception thrown: 'System.Threading.ThreadAbortException' in
CSharpLabProblems. Worker thread: Exiting.
Main thread: Thread lifecycle management completed.
```

13.2 Thread Naming and Priority

Create multiple threads, assigns names to each thread, and sets different priorities. Display the thread names and priorities during execution.

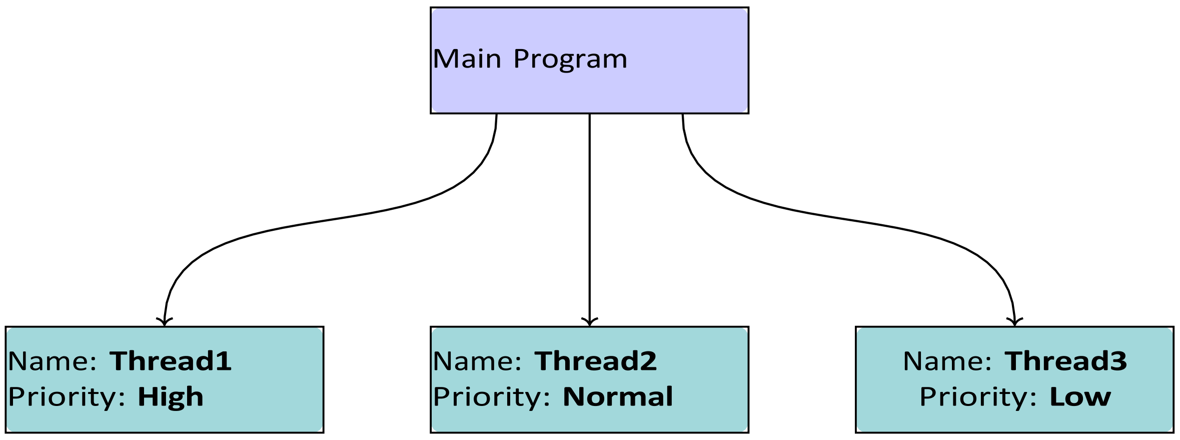


Figure 13.1: Thread naming and priorities

Instructions:

1. Create a list of tasks to be executed by different threads.
2. Assign a unique name to each thread.
3. Set different priority levels (High, Normal, Low) for each thread.
4. Start all threads and display their names and priorities during execution.
5. Ensure all threads complete their tasks.

Output:

```
Thread Name: Thread1, Priority: High, Status:
Running Thread Name: Thread2, Priority: Normal,
Status: Running Thread Name: Thread3, Priority:
Low, Status: Running
```

13.3 Multiple Threads

Develop a program that demonstrates the lifecycle management of multiple threads, including starting each thread, waiting for them to complete their work, and printing specific messages at different stages of their execution.

Program Requirements

1. Create three separate threads, each performing a simulated task (e.g., printing messages, performing calculations).
2. Print messages when each thread starts its work and when each thread completes its task.
3. Ensure the main thread waits for all threads to finish their tasks before proceeding

further.

4. Display a message indicating when all threads have completed their tasks.

Output

The program should output messages in the following sequence:

```
Thread        1:
Starting
work...   Thread
2:      Starting
work...   Thread
3:      Starting
work...
Main thread: Waiting for all threads to
complete... Thread 1: Work completed.
Thread 2: Work
completed.
Thread 3: Work
completed.
Main thread: All threads have completed.
```

13.4 Thread Synchronization with Mutex

Develop an application where multiple threads increment a shared counter. Use a mutex to synchronize access to the counter and ensure thread safety.

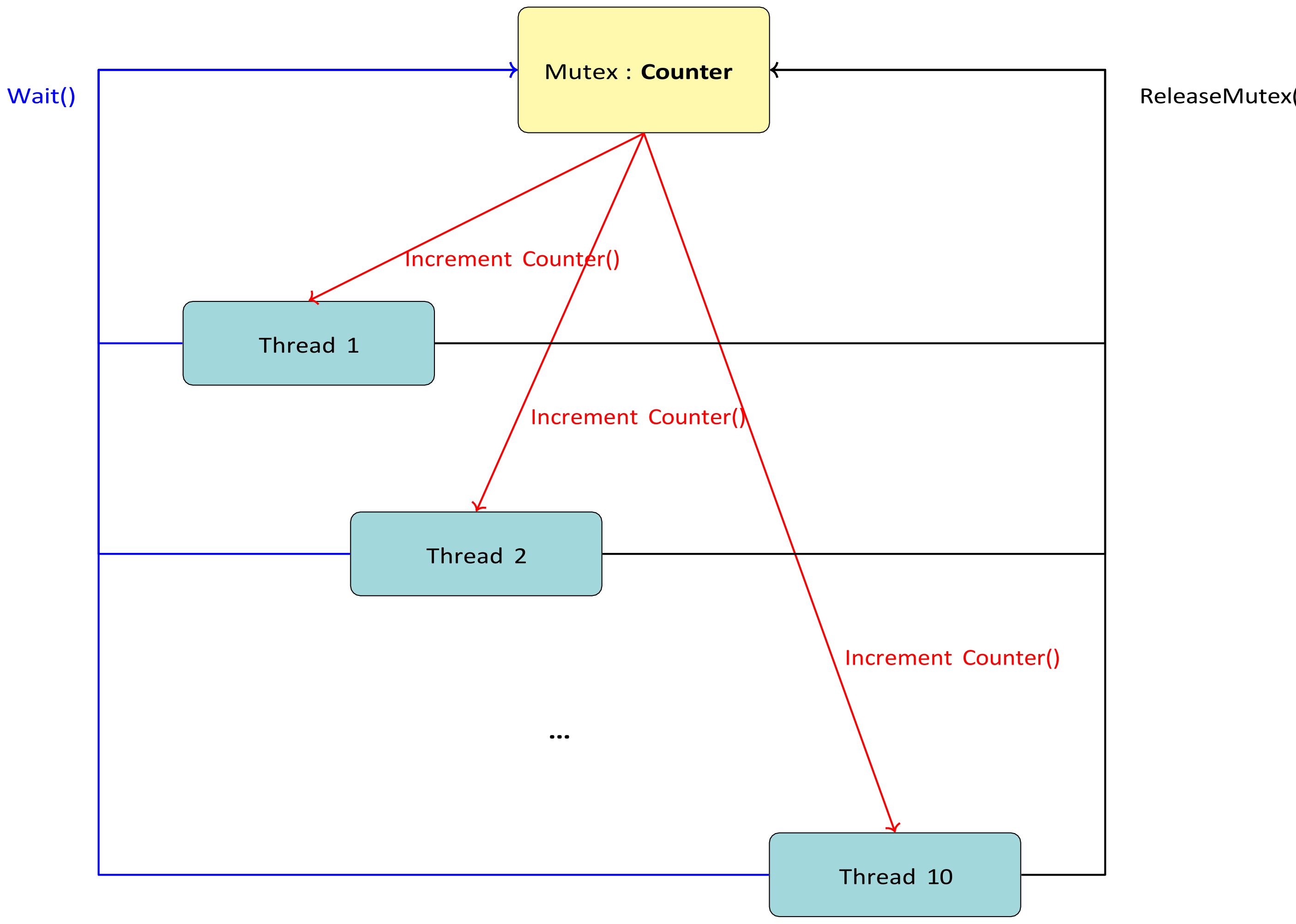


Figure 13.2: Thread synchronization with mutex, illustrating Wait(), Increment Counter, and ReleaseMutex() actions

Instructions:

Create a shared counter variable.

Create multiple threads that increment the counter.

Use a mutex to ensure only one thread can increment the counter at a time.

Start all threads and wait for their completion.

Display the final value of the counter.

Output:

```
Final Counter Value: 1000
```

13.5 Producer-Consumer Problem

Implement the producer-consumer problem using threads. Use condition variables to synchronize producer and consumer actions.

Instructions:

- Create a shared buffer (queue).
- Implement a producer thread that generates items and adds them to the buffer.
- Implement a consumer thread that removes and processes items from the buffer.
- Use condition variables to notify the consumer when new items are available and the producer when there is space in the buffer.
- Ensure proper synchronization to avoid race conditions.

Output:

```
Producer: Produced item 1
Buffer is full. Waiting for
consumer... Consumer: Consumed
item 1
Buffer is empty. Waiting for
producer... Producer: Produced
item 2
Buffer is full. Waiting for
consumer... Consumer: Consumed
item 2
Buffer is empty. Waiting for
producer... Producer: Produced
item 3
Buffer is full. Waiting for
consumer... Consumer: Consumed
item 3
...
```

13.6 Thread Pool Implementation

Design a thread pool manager that manages a fixed number of worker threads. Implement task submission and ensure tasks are executed concurrently.

Instructions:

- Create a thread pool manager with a fixed number of threads.
- Implement a method to submit tasks to the thread pool.
- Ensure tasks are executed concurrently by available threads in the pool.
- Use synchronization techniques to manage task execution and completion.
- Display the status of tasks and thread pool utilization.

Output:

```
Task 1 started by Thread Thread 1.
Task 2 started by Thread Thread 2.
Task 2 completed by Thread Thread 2.
Task 1 completed by Thread Thread 1.
Task 3 started by Thread Thread 2.
Task 3 completed by Thread Thread 2.
```

Mini Projects

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

14.1 Desktop Applications

1. **Student Management System:** Manage student records, courses, grades, and attendance.
2. **Inventory Management System:** Track inventory levels, orders, and sales.
3. **Library Management System:** Manage books, patrons, and borrowing records.
4. **Hospital Management System:** Manage patient records, appointments, and medical history.
5. **Employee Management System:** Manage employee information, payroll, and leave applications.
6. **Task Management System:** Track tasks, assignees, and project deadlines.
7. **Chat Application:** Implement real-time messaging between users or groups.

14.2 Web Applications

1. **Online Quiz Application:** Create quizzes with multiple-choice questions.
2. **E-commerce Website:** Develop a fully functional online store with product listings and shopping cart.
3. **Blog or Content Management System:** Manage blog posts, articles, and multimedia content.
4. **Customer Relationship Management (CRM) System:** Manage customer interac- tions, sales leads, and support tickets.
5. **Personal Finance Manager:** Track income, expenses, budgets, and savings goals. On- line Learning Platform: Host courses, quizzes, and educational resources.

14.3 Mobile Applications

1. **Fitness Tracker:** Track workouts, set goals, and monitor progress.
2. **Recipe Manager:** Organize recipes, create shopping lists, and meal plans.
3. **Expense Tracker App:** Track expenses, categorize spending, and set budgets.
4. **Language Learning App:** Provide lessons, quizzes, and interactive exercises for learning languages.
5. **Tourist Guide App:** Offer information on attractions, maps, and local recommendations.

14.4 Game Development

1. **2D Platformer Game:** Create a classic side-scrolling platform game with levels and obstacles.
2. **Puzzle Game:** Design and implement various puzzles with logic challenges and levels.
3. **RPG (Role-Playing Game):** Develop a role-playing game with quests, characters, and combat mechanics.
4. **Simulation Game:** Build a simulation game around a specific theme like city-building or farming.
5. **Card Game:** Implement a card game with rules, multiplayer support, and AI opponents.

14.5 IoT Applications

1. **Smart Home Control System:** Create an application to control IoT devices at home (lights, temperature, security cameras).
2. **Environmental Monitoring System:** Develop a system to collect and display data from sensors (temperature, humidity, air quality).
3. **Health Monitoring Device Integration:** Integrate wearable health devices (like fitness trackers) to track and display health metrics.
4. **Industrial IoT Dashboard:** Build a dashboard to monitor and manage industrial equip- ment and processes.
5. **Smart Agriculture System:** Develop an IoT solution for monitoring soil moisture, temperature, and crop health.

14.6 Data Analysis and Reporting

1. **Dashboard for Business Analytics:** Create a dashboard to visualize sales data, trends, and key performance indicators (KPIs).
2. **Real-time Data Streaming Application:** Build an application to process and visualize real-time data streams (IoT sensor data, social media feeds).
3. **Predictive Analytics Tool:** Develop a tool to analyze historical data and make predic- tions using machine learning algorithms.
4. **Financial Portfolio Tracker:** Track investments, analyze portfolio performance, and provide recommendations.
5. **Healthcare Data Visualization:** Visualize patient data, medical trends, and health outcomes for healthcare professionals.

14.7 Guidelines for Mini Projects

This section provides guidelines for students to undertake various mini projects such as web applications, mobile applications, gaming applications, IoT applications, and data mining ap- plications. These guidelines include the necessary knowledge and tools required for each project type, considering that students have covered up to multi-threading in their course.

14.7.1 Desktop Applications

Knowledge Required:

- Basics of desktop application development
- Understanding of Windows Forms and WPF (Windows Presentation Foundation)
- Event-driven programming
- Multi-threading for responsive UI
- Data binding and MVVM (Model-View-ViewModel) architecture

Tools and Technologies:

- Visual Studio
- .NET Framework or .NET Core
- Windows Forms or WPF
- Git for version control

Guidelines:

- Start by designing the user interface using Windows Forms or WPF.
- Implement event handlers for user interactions (e.g., button clicks, form inputs).
- Use data binding to connect UI elements to data sources.
- Apply the MVVM pattern for better separation of concerns and maintainability.
- Implement multi-threading to ensure the UI remains responsive during long-running tasks.
- Test the application thoroughly for functionality and performance.

- Ensure the application handles exceptions and provides appropriate error messages to users.

14.7.2 Web Applications

Knowledge Required:

- Basics of web development (HTML, CSS, JavaScript)
- C# and ASP.NET Core for backend development
- Understanding of MVC (Model-View-Controller) architecture
- Working with databases (SQL Server, Entity Framework)
- Asynchronous programming with async and await

Tools and Technologies:

- Visual Studio or Visual Studio Code
- ASP.NET Core
- SQL Server or other relational databases
- Git for version control

Guidelines:

- Start by designing the database schema.
- Create the backend using ASP.NET Core and Entity Framework.
- Develop the frontend using HTML, CSS, and JavaScript.
- Integrate frontend and backend using MVC architecture.
- Implement asynchronous operations for improved performance.
- Test the application thoroughly before deployment.

14.7.3 Mobile Applications

Knowledge Required:

- Basics of mobile development (Xamarin for C#)
- Understanding of mobile UI/UX design
- Knowledge of RESTful services for backend integration
- Asynchronous programming for handling background tasks

Tools and Technologies:

- Visual Studio with Xamarin
- RESTful APIs
- SQLite for local data storage
- Git for version control

Guidelines:

- Design the mobile application's UI/UX.
- Use Xamarin.Forms for cross-platform development.
- Integrate with RESTful APIs for backend data.

- Implement local data storage using SQLite.
- Use asynchronous programming to handle network calls.
- Test the application on multiple devices and screen sizes.

14.7.4 Gaming Applications

Knowledge Required:

- Basics of game development (Unity with C#)
- Understanding of game physics and animations
- Multi-threading for handling game loops and rendering

Tools and Technologies:

- Unity Engine
- Visual Studio or Visual Studio Code
- Git for version control

Guidelines:

- Start by designing the game concept and mechanics.
- Use Unity to develop the game environment and characters.
- Implement game logic and physics using C# scripts.
- Utilize multi-threading to optimize game performance.
- Test the game thoroughly for bugs and performance issues.

14.7.5 IoT Applications

Knowledge Required:

- Basics of IoT (Internet of Things) concepts
- Working with sensors and microcontrollers (e.g., Arduino, Raspberry Pi)
- Networking basics for IoT device communication
- Asynchronous programming for sensor data handling

Tools and Technologies:

- Arduino IDE or Raspberry Pi setup
- Visual Studio for C# development
- MQTT or HTTP for IoT communication
- Git for version control

Guidelines:

- Start by designing the IoT system architecture.
- Connect and configure sensors and microcontrollers.
- Develop the C# application to process and display sensor data.
- Implement communication protocols (MQTT/HTTP) for data transmission.
- Use asynchronous programming for real-time data handling.

- Test the IoT system in different environments and conditions.

14.7.6 Data Mining Applications

Knowledge Required:

- Basics of data mining and machine learning
- Understanding of data preprocessing and cleaning
- C# libraries for data mining (e.g., ML.NET)
- Multi-threading for handling large datasets

Tools and Technologies:

- Visual Studio with ML.NET
- SQL Server or other data storage solutions
- Git for version control

Guidelines:

- Start by collecting and preprocessing the dataset.
- Use ML.NET to develop data mining models.
- Implement the models in a C# application.
- Use multi-threading to process large datasets efficiently.
- Evaluate and validate the data mining results.
- Test the application with different datasets and scenarios.

V. TEXT BOOKS:

1. Troelsen, A., & Japikse.“Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming Eleventh Edition”, Apress, Eleventh edition,2022.
2. Jon Skeet ”C# in Depth” Fourth Edition, Manning, 2019.

VI. REFERENCE BOOK

1. Andrew Stellman,Jennifer Greene. ”Head First C#: A Learner’s Guide to Real-World Programming with C# and .NET ”, O’Reilly Media, 5th Edition,2024.
2. Ian Griffiths. ”Programming C# 10: Build Cloud, Web, and Desktop Applications”, O’Reilly Media, First Edition, 2022.
3. Mark J Price. ”C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals”, Packt Publishing, 8th edition,2023.

VII. ELECTRONICS RESOURCES

1. https://learn.microsoft.com/en-us/dotnet/csharp
2. https://www.codecademy.com/learn/learn-c-sharp
3. https://www.pluralsight.com/paths/c-10
4. https://www.c-sharpcorner.com/
5. https://www.dotnetperls.com/category c

VIII. MATERIALS ONLINE

1. Course template
2. Lab manual

EXPERIENTIAL ENGINEERING EDUCATION (EXEED) – PROTOTYPE / DESIGN BUILDING

III Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD12	**Skill**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	2	1	40	60	100
Contact Classes: NIL	**Tutorial Classes: NIL**	**Practical Classes:45**				**Total Classes: 45**		
Prerequisite: Essentials of Innovation.								

I. COURSE OVERVIEW:

This course provides an overall exposure to the various methods and tools of prototyping. This course discusses Low- Fidelity, paper, wireframing and tool-based prototyping techniques along with design principles and patterns.

II. COURSES OBJECTIVES:

The students will try to learn

I. The basic principles and design aspect of prototyping.
II. The various techniques, design guidelines and patterns.
III. The applications of prototyping using various tools and platforms.

III. SYLLABUS:

Module 1: An Introduction to Prototyping

Objective: Understand the basics of prototyping and its importance in the design process.
Exercise:

1. Introduction Lecture (10 minutes): Overview of prototyping and its types.
2. Brainstorming Session (20 minutes): Brainstorm simple app ideas.
3. Sketching Session (20 minutes): Create initial design sketches on paper.
4. Paper Prototyping (40 minutes): Build a low-fidelity paper prototype of the app’s user interface.
5. Presentation and Feedback (20 minutes): Present prototypes and receive feedback.
   Materials and Tools: Paper, pencils, rulers, scissors, glue.

Module 2: Low-Fidelity Prototyping and Paper Prototyping

Objective: Learn concepts and techniques of low-fidelity and paper prototyping.
Exercise:

1. Lecture (10 minutes): Concepts of low-fidelity prototyping.
2. Hands-On Session (40 minutes): Build paper prototypes of the app’s screens and user flow.

3. Peer Review (30 minutes): Exchange prototypes with peers for feedback.
4. Iteration (30 minutes): Refine prototypes based on feedback.

Materials and Tools: Paper, pencils, rulers, scissors, glue.

Module 3: Wireframing and Tool-Based Prototyping

Objective: Create wireframes and digital prototypes using software tools.
Exercise:

1. Lecture (10 minutes): Introduction to wireframing and digital prototyping tools.
2. Wireframing (30 minutes): Design wireframes of the app's user interface using tools like Figma or Sketch.
3. Tool-Based Prototyping (50 minutes): Create a digital prototype of the app.
4. Presentation and Feedback (20 minutes): Share digital prototypes and gather feedback.
   Materials and Tools: Wireframing tools, computers.

Module 4: Physical Low-Fidelity Prototyping and 3D Printing

Objective: Build simple physical prototypes using basic materials and introduce 3D printing.
Exercise:

1. Lecture (10 minutes): Importance of physical prototyping.
2. Prototype Building (30 minutes): Create a physical representation of the app's main interface using materials like cardboard and foam.
3. **Introduction to 3D Printing (20 minutes):** Overview of 3D printing technology and its applications in prototyping.
4. Testing (20 minutes): Test the physical prototypes for basic functionality.
5. Feedback Session (30 minutes): Present and discuss prototypes.

Materials and Tools: Cardboard, foam, clay, scissors, glue, 3D printer.

Module 5: Tool-Based Prototyping and Circuit Design

Objective: Develop advanced digital prototypes using specialized tools and introduce basic circuit design concepts.
Exercise:

1. Lecture (10 minutes): Advanced prototyping tools and techniques, introduction to circuit design.
2. Hands-On Session (40 minutes): Use tools like Figma, Adobe XD, or Android Studio to create an interactive prototype of the app.
3. **Circuit Design (20 minutes):** Design simple circuits that could be integrated into the app's hardware prototype.
4. User Testing (20 minutes): Conduct user tests and document feedback.
5. Feedback and Iteration (20 minutes): Refine prototypes based on user feedback.

Materials and Tools: Advanced prototyping software, computers, circuit design tools (e.g., breadboards, resistors, LEDs).

Module 6: Design Principles and Patterns - Graphic Design

Objective: Apply graphic design principles to create visually appealing app interfaces.
Exercise:

1. Lecture (10 minutes): Graphic design principles (contrast, alignment, repetition, proximity).
2. Design Session (50 minutes): Create a graphic design layout for the app's interface.
3. Peer Review (30 minutes): Exchange designs for feedback.
4. Refinement (20 minutes): Refine designs based on peer feedback.

Materials and Tools: Graphic design software, computers.

Module 7: Interaction Design and Arduino Integration

Objective: Apply interaction design principles to create intuitive app interfaces and introduce Arduino for hardware interactions.
Exercise:

1. Lecture (10 minutes): Basics of interaction design and Arduino integration.
2. Design Session (40 minutes): Develop interactive wireframes or prototypes for the app's user interactions.
3. **Arduino Integration (20 minutes):** Connect simple Arduino circuits to the app prototype to simulate interactions (e.g., button presses, LED responses).
4. User Testing (20 minutes): Conduct usability tests with peers.
5. Iteration (20 minutes): Improve designs based on test results.

Materials and Tools: Interaction design tools, computers, Arduino kits.

Module 8: Commercial Design Guidelines and Standards

Objective: Design according to industry standards and guidelines.
Exercise:

1. Lecture (10 minutes): Overview of commercial design guidelines
2. Design Session (50 minutes): Develop a prototype of the app following specific design guidelines.
3. Review (30 minutes): Evaluate prototypes against the guidelines.
4. Presentation and Discussion (20 minutes): Present prototypes and discuss adherence to guidelines.

Materials and Tools: Design guidelines documents, design software, computers.

Module 9: Universal Design: Sensory and Cognitive Impairments

Objective: Create app designs accessible to users with sensory and cognitive impairments.
Exercise:

1. Lecture (10 minutes): Principles of universal design.
2. Design Session (50 minutes): Develop an accessible design for the app.
3. User Testing (20 minutes): Test designs with peers simulating impairments.
4. Feedback Session (30 minutes): Present and discuss accessibility features.

Materials and Tools: Design software, computers, accessibility guidelines.

Module 10: Universal Design: Tools, Limitations, and Standards

Objective: Understand tools and limitations in universal design, and adhere to standards.
Exercise:

1. Lecture (10 minutes): Tools and limitations in universal design.
2. Hands-On Session (50 minutes): Use tools to create universally designed app prototypes.
3. Testing (20 minutes): Evaluate prototypes for accessibility and adherence to standards.
4. Iteration (30 minutes): Refine designs based on testing outcomes.

Materials and Tools: Universal design tools, computers.

Module 11: Mobile UI Design, CNC, and Arduino

Objective: Design user interfaces for mobile and wearable devices, create related hardware prototypes using CNC tools, and integrate Arduino for interactions.
Exercise:

1. Lecture (10 minutes): Principles of mobile UI and wearable design, introduction to CNC lathe and mill, and Arduino for wearable interactions.
2. Design Session (30 minutes): Develop a UI prototype for a mobile app or wearable device.
3. **CNC Prototyping (20 minutes):** Create physical components of the wearable device using CNC lathe and mill.
4. **Arduino Integration (20 minutes):** Program Arduino for basic interactions (e.g., motion detection, feedback via LEDs).
5. User Testing (20 minutes): Conduct usability tests on the prototype.
6. Feedback Session (30 minutes): Present and discuss usability findings.

Materials and Tools: Mobile UI design tools, computers, mobile devices, CNC lathe, CNC mill, Arduino kits.

Module 12: Automotive User Interface, Laser Engraving, and Arduino Integration

Objective: Design user interfaces for automotive applications, integrate laser engraving for UI elements, and use Arduino for sensor-based interactions.
Exercise:

1. Lecture (10 minutes): Fundamentals of automotive UI design, overview of laser engraving, and Arduino for sensor integration.
2. Design Session (30 minutes): Create a prototype for an automotive interface (e.g., dashboard layout).
3. **Laser Engraving (20 minutes):** Use laser engraving to create high-precision UI elements for the automotive interface.
4. **Arduino Integration (20 minutes):** Implement Arduino to simulate sensor-based interactions (e.g., temperature control, obstacle detection).
5. Testing (20 minutes): Test the prototype for usability and functionality.
6. Feedback Session (30 minutes): Present and discuss design improvements.

Materials and Tools: Automotive UI design tools, computers, laser engraving machine, Arduino kits.

IV. REFERENCE BOOKS:

1. Chee Kai Chua, Kah Fai Leong, "3D Printing and Additive Manufacturing: Principles and Applications".
2. Simon Knight, "Arduino for Beginners: Step-by-Step Guide to Arduino (Arduino Hardware & Software)".
3. Peter Smid, "CNC Programming Handbook".
4. Steven Wolfe, "Laser Cutting and Engraving".
5. Charles Platt, "Make: Electronics: Learning Through Discovery".

COURSE CONTENT

DESIGN AND ANALYSIS OF ALGORITHMS								
IV Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD13	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisite: **Programming for Problem Solving, Data Structures**								

I. COURSE OVERVIEW:

Design and analysis of algorithms is the process of finding the computational complexity of algorithms. It helps to design and analyze the logic on how the algorithm will work before developing the actual code for a program. It focuses on introduction to algorithm, asymptotic complexity; sorting and searching using divide and conquer, greedy method, dynamic programming, backtracking, branch and bound. NP-hard and NP-complete problems. The applications of algorithm design are used for information storage, retrieval, transportation through networks, and presentation to users.

II. COURSES OBJECTIVES:

The students will try to learn

I. Mathematical approach for Analysis of Algorithms.
II. Methods and techniques for analyzing the correctness and resource requirements of algorithms.
III. Different paradigms of algorithm design including recursive algorithms, divide-and-conquer algorithms, dynamic programming, greedy algorithms, Backtracking, Branch and Bound and graph algorithms.
IV. Strategies for solving problems not solvable in polynomial time.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Find the (worst case, randomized, amortized) running time and space complexity of given algorithms using techniques such as recurrences and properties of probability.
CO2	Apply divide and conquer algorithms for solving sorting, searching and matrix multiplication.
CO3	Make Use of appropriate tree traversal techniques for finding shortest path.
CO4	Compare Identify suitable problem solving techniques for a given problem and finding optimized solutions using Greedy and Dynamic Programming techniques
CO5	Apply greedy algorithm Utilize backtracking and branch and bound techniques to deal with traceable and in-traceable problems.
CO6	Apply Describe the classes P, NP, NP-Hard, and NP- complete for solving deterministic and non-deterministic problems.

IV. COURSE CONTENT:

MODULE – I: INTRODUCTION (10)

Algorithm: Pseudo code for expressing algorithms; Performance analysis: Space complexity, time complexity; Asymptotic notations: Big O notation, omega notation, theta notation and little o notation, amortized complexity; Divide and Conquer: General method, binary search, quick sort, merge sort, Stassen’s matrix multiplication.

MODULE – II: SEARCHING AND TRAVERSAL TECHNIQUES (09)

Disjoint set operations, union and find algorithms; Efficient non recursive binary tree traversal algorithms, spanning trees; Graph traversals: Breadth first search, depth first search, connected components, biconnected components.

MODULE – III: GREEDY METHOD AND DYNAMIC PROGRAMMING (10)

Greedy method: The general method, job sequencing with deadlines, knapsack problem, minimum cost spanning trees,single source shortest paths.

Dynamic programming: The general method, matrix chain multiplication optimal binary search trees, 0/1 knapsack problem, single source shortest paths, all pairs shortest paths problem, the travelling salesperson problem.

MODULE – IV: BACKTRACKING AND BRANCH AND BOUND (09)

Backtracking: The general method, the 8 queens problem, sum of subsets problem, graph coloring, Hamiltonian cycles; Branch and bound: The general method, 0/1 knapsack problem, least cost branch and bound solution, first in first out branch and bound solution, travelling salesperson problem.

MODULE – V: NP-HARD AND NP-COMPLETE PROBLEMS (09)

Basic concepts: Non-deterministic algorithms, the classes NP - Hard and NP, NP Hard problems, clique decisionproblem, chromatic number decision problem, Cook's theorem.

V. TEXT BOOKS:

1. Ellis Horowitz, Satraj Sahni, Sanguthevar Rajasekharan, “Fundamentals of Computer Algorithms”, Universities Press, 2nd edition, 2015.
2. Alfred V. Aho, John E. Hopcroft, Jeffrey D, “The Design And Analysis Of Computer Algorithms”, Pearson India, 1st edition, 2013..

VI. REFERENCE BOOKS:

1. Levitin A, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, 3rd edition, 2012.
2. Goodrich, M. T. R Tamassia, “Algorithm Design Foundations Analysis and Internet Examples”, John Wileyn and Sons, 1st edition, 2001.
3. Base Sara Allen Vangelder, “Computer Algorithms Introduction to Design and Analysis”, Pearson, 3rd edition, 1999.

VII. ELECTRONICS RESOURCES:

1. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
2. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
3. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

WEB SYSTEMS ENGINEERING									
IV Semester: CSE / IT									
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	T	P	C	CIA	SEE	Total
ACSD14	Core	3	0	0	3	40	60	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		
Prerequisite: Object Oriented Programming								

I. COURSE OVERVIEW:

This course introduces students to create concurrently a web app and a native app (for Android and iOS) with React Native and React Native Web. It covers HTML for structuring and presenting content on the World Wide Web. CSS being used to format structured content. To create a dynamic and interactive experience for the user it covers JAVASCRIPT. To build the applications using React concepts such as JSX, REDUX and PHP.

II. COURSESOBJECTIVES:

The students will try to learn

I. The fundamentals of HTML and CSS to design static and dynamic web pages.
II. The concepts of client side programming with Bootstrap , JavaScript, Ajax , JSX.
III. The MVC architecture, about React and built single and multiple page applications using REACT with REDUX.
IV. The characteristics, systematic methods, model for developing web applications using PHP.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Summarize HTML elements and attributes for structuring and presenting content of webpage based on the user requirement.
CO 2	Make use of CSS properties for formatting webpages.
CO 3	Develop responsive webpage using Bootstrap for viewing web pages in various devices
CO 4	Utilize the concepts of JS with event actions for displaying information on webpages.
CO 5	Identify UI binding library elements for deploying a reusable complex UI React Redux.
CO 6	Develop web applications with the help of React Redux framework and serverside scripting using PHP.

IV.COURSE CONTENT:

MODULE - I: INTRODUCTION TO WEB APPLICATION, HTML AND CSS (10)

Introduction to web application: Internet, Intranet, WWW, Static and Dynamic Web Page; Web Clients; Web Servers; Basics of HTML5 and web design, creating tables, lists, HTML forms. Styles and classes to your web pages, web page layouts with CSS, introduction to responsive web design with CSS3 and HTML5.

MODULE - II: BUILD INTERFCAES USING BOOTSTARP (09)

Introduction to web design from an evolutionary perspective, user interface design through bootstrap, containers, tables, jumbotrons, list, cards, carousal, navigation, modals, flex and forms, responsive web page design, basic UI grid structure.

MODULE - III: INTERACTIVE USER INFERFACE AND WEB APPLICATION DEVELOPMENT (10)

JavaScript variable naming rules, data types, expressions and operators, pattern matching with regular

expressions, managing web page styles using JavaScript and CSS, script forms, introduction to AJAX.
Introduction to web design from an evolutionary perspective, create a native and web app, JSX, class and function components, props, state, lifecycle methods, and hooks.

MODULE - IV: UI BINDING LIBRARY FOR REACT AND REDUX (09)

Introduction to client-side routing using React Router, global state management and transitions using REDUX, server -side rendering and testing using Jest. Web Development Using REACT is delivered both in a blended learning and self-paced mode.
REDUX store using the official create store function, REDUX toolkit has a configure store API, loading state for that particular API, adding an API service as a middleware, example uses create REACT App.

MODULE - V: INTRODUCTION TO PHP AND SERVER-SIDE SCRIPTING (10)

Introduction to PHP: Declaring variables, data types, arrays, strings, operations, expressions, control structures, functions, Reading data from web form controls like Text Boxes, radio buttons, lists etc., Handling File Uploads, Connecting to database (My SQL as reference), executing simple queries, handling results, Handling sessions and cookies.
File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

V. TEXT BOOKS:

1. Alok Ranjan Abhilasha Sinha, Ranjit Battewad, "JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript", 1st edition, 2020
2. Alex Banks and Eve Porcello, "Learning React: Functional Web Development with React and Redux", 2017.
3. The Complete Reference PHP – Steven Holzner, Tata McGraw-Hill, 2017.

VI. REFERENCE BOOKS:

1. Adam Boduch and Roy Derks, "React and React Native: A complete hands-on guide to modern web and mobile development with React.js", 3rd edition, 2020.
2. W Hans Bergsten, "Java Server Pages", O'Reilly, 3rd edition, 2003
3. D.Flanagan, "Java Script", O'Reilly, 6th edition, 2011.
4. Jon Duckett, "Beginning Web Programming", WROX, 2nd edition, 2008.

VII. ELECTRONICS RESOURCES:

1. http://computer.howstuffworks.com/computer-networking-channel.htm
2. https://www.geeksforgeeks.org/layers-osi-model/
3. https://www.wikilectures.eu/w/Computer_Network
4. https://technet.microsoft.com/en-us/network/default.aspx

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

OBJECT ORIENTED SOFTWARE ENGINEERING								
IV Semester: CSE / IT / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD15	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 48	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 48**		
Prerequisite: There is no prerequisite to take this course								

I. COURSE OVERVIEW:

This course concentrates on developing basic understanding about various activities that are involved in a software development. This course enables the student to develop necessary skills for developing a product or applications. The course focuses on all activities involved in software development (communication, planning, modelling, construction, deployment).In this course; students will gain a broad understanding of the discipline of software engineering and its application to the development and management of software systems. Student can implement and get knowledge about development of the software and gains knowledge of basic engineering methods and practices, and their appropriate application

II. COURSES OBJECTIVES:

The students will try to learn

I. The concepts of object-oriented programming in software design and development.
II. The different phases in software development life cycle.
III. The Design concepts in software development using unified modelling language.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Identify the appropriate process model approaches and techniques to manage a given software development process.
CO2	Summarize the software requirement specifications and the SRS documents.
CO3	Identify various modelling approaches for Object Oriented Analysis using UML.
CO4	Understand the importance and need of Design and explain the various approaches of Designs.
CO5	Explain various Testing Approaches and automation tools
CO6	Explain the importance of Software Maintenance and Reengineering process

IV. COURSE CONTENT:

MODULE –I: INTRODUCTION TO SOFTWARE ENGINEERING (09)

Introduction to software engineering, software development process models, agile development, project and process, project management, process and project metrics, object-oriented concepts, principles and methodologies.

MODULE –II: PLANNING AND SCHEDULING (10)

Software requirements specification, software prototyping, software project planning, scope, resources, software estimation, empirical estimation models, planning, risk management, software project scheduling, object-oriented estimation and scheduling.

MODULE –III: ANALYSIS (10)

Analysis modeling, data modeling, functional modeling and information flow, behavioural modeling, structured analysis, object-oriented analysis, domain analysis.

Object-oriented analysis process, object relationship model, object behaviour model, design modeling with UML.

MODULE –IV: DESIGN (09)
Design concepts and principles, design process, design concepts, modular design, design effective modularity, introduction to software architecture, data design, transform mapping, transaction mapping, object-oriented design, system design process, object design process.

MODULE –V: IMPLEMENTATION, TESTING AND MAINTENANCE (10)
Top-down, bottom-up, object-oriented product implementation and integration. Software testing methods, white box, basis path, control structure, black box, unit testing, integration testing, validation and system testing, testing tools, software maintenance and reengineering.

V. TEXT BOOKS:

1. Ivar Jacobson, "Object Oriented Software Engineering: A Use Case Driven Approach", Pearson India, 1st edition, 2002.
2. Bernd Bruegge, Allen H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns and Java", Pearson New International Edition, 3rd edition, 2013.

VI. REFERENCE BOOKS:

1. Roger. S. Pressman and Bruce R. Maxim, "Software Engineering – A Practitioner's Approach", McGraw Hill, 7th edition, 2015.
2. Ian Sommerville, "Software Engineering", Pearson Education, New Delhi, 8th edition, 2011.
3. Bill Brykczynski, Richard D. Stutz, "Software Engineering- Project Management", Wiley-IEEE Computer Society, 2nd edition, 2000.
4. Craig Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", Pearson Education, 3rd edition, 2008.
5. Jalote P, "An Integrated Approach to Software Engineering", Narosa Publishers, New Delhi, 3rd edition 2013.

VII. ELECTRONICS RESOURCES:

1. https://onlinecourses.nptel.ac.in/noc19_cs52/preview
2. https://ece.iisc.ac.in/~parimal/2019/ml.html
3. http://www.springer.com/gp/book/9780387848570"www.springer.com/gp/book/9780387848570
4. http://www.cse.iitb.ac.in/~sunita/cs725/calendar.html"www.cse.iitb.ac.in/~sunita/cs725/calendar.html
5. https://cs.nyu.edu/~mohri/mlu11/

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

DATABASE MANAGEMENT SYSTEMS								
IV Semester: CSE /IT / CSE (AI&ML) / CSE (DS) / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AITD03	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 45	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving, Data Structures								

I. COURSE OVERVIEW:

The purpose of this course is to provide a clear understanding of fundamentals with emphasis on their applications to create and manage large data sets. It highlights on technical overview of database software to retrieve data from n database. The course includes database design principles, normalization, concurrent transaction processing,, security, recovery and file Organization techniques.

II. COURSES OBJECTIVES:

The students will try to learn

I. The role of database management system in an organization and learn the data base concepts.
II. The design of databases using data modeling and Logical database design techniques.
III. The database queries using relational algebra and calculus and SQL.
IV. The concept of a database transaction and related concurrent, recovery facilities and evaluate a set of queries in query processing.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Describe data models, schemas, instances, view levels and database architecture for voluminous data storage.
CO2	Define the concept of Relational Algebra and Relational Calculus from set theory to represent queries.
CO3	Make Use of SQL queries for data aggregation, calculations, views, sub-queries, embedded queries manipulation.
CO4	Illustrate the definition of Functional Dependencies, Inference rules and minimal sets of FD's to maintain data integrity.
CO5	State the concepts of transaction, states and ACID properties in data manipulation.
CO6	Apply indexing, hashing techniques to access the records from the file effectively.

IV. COURSE CONTENT:

MODULE–I:CONCEPTUALMODELINGINTRODUCTION (10)

Introduction to Databases: Purpose of Database systems, view of data, data models, Database languages, Database users, various components of overall DBS architecture, various concepts of ER model, basics of Relational Model.

MODULE–II:RELATIONALAPPROACH (09)

Relational algebra and calculus: Relational algebra, selection and projection, set operations, renaming, joins, division, examples of algebra queries, relational calculus: Tuple relational calculus, Domain relational calculus, expressive power of algebra and calculus.

MODULE–III:SQLQUERY-BASICS,RDBMS- NORMALIZATION (10)

SQL – Data Definition commands, Queries with various options, Mata manipulation commands, Views, Joins, views, integrity and security, triggers and cursors;

Relational database design: Pitfalls of RDBD, Lossless join decomposition, functional dependencies, Armstrong axioms, normalization for relationaldatabases1st, 2nd and 3rd normal forms, Basic definitions of MVDs and JDs, 4th and 5th normal forms.

MODULE–IV:TRANSACTIONMANAGEMENT (09)
Transaction processing: Transaction concept, transaction State, implementation of atomicity and durability, concurrent executions, serializability, recoverability.

Concurrency Control: Lock-based protocols, timestamp-based protocols, validation-based protocols, multiple granularity, multiversion schemes, deadlock handling.
Recovery: Failure classification, storage structure, recovery and atomicity, Log-Based recovery, shadow paging, recovery with concurrent transactions buffer management.

MODULE–V:DATASTORAGEANDQUERYPROCESSING (10)
Data storage: Overview of physical storage media, magnetic disks, storage access, file organization, organization of records in files.

Indexing and Hashing: Basic concepts: Ordered indices, B+ tree index files, B-tree index files, static hashing, Dynamic Hashing, Comparison of Ordered Indexing and Hashing.
Query Processing: Overview, measures of query cost.

V. TEXT BOOKS:

1. Abraham Silberschatz, Henry F.Korth, S.Sudarshan, “Database System Concepts", McGraw-Hill, 6th edition, 2017.

VI. REFERENCE BOOKS:

1. Ramez Elmasri, Shamkant, B. Navathe, “Database Systems”, Pearson Education, 6th edition, 2013.
2. Peter Rob, Carles Coronel, “Database System Concepts”, Cengage Learning, 7th edition, 2008.

VII. ELECTRONICS RESOURCES:

1. https://www.youtube.com/results?search_query=DBMS+onluine+classes
2. http://www.w3schools.in/dbms/
3. http://beginnersbook.com/2015/04/dbms-tutorial/

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

COMPUTER NETWORKS								
II Semester: CSE / CSE (AI & ML) / CSE (CS) / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AITD04	Core	L	T	P	C	CIA	SEE	Total
		3	0	0	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite: There is no prerequisite to take this course								

I. COURSE OVERVIEW:

The main emphasis of this course is on the organization and management of local area networks (LANs) wide area networks (WANs). The course includes learning about computer network organization and implementation, obtaining a theoretical understanding of data communication and computer networks. Topics include layered network architectures, addressing, naming, forwarding, routing, communication reliability, the client-server model, and web and email protocols. The applications of this course are to design, implement and maintain basic computer networks.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts of computer networking, different types of topologies, and protocols
II. Error correction and detection examples and applications in data link layer and uses of other media access control.
III. The data transmission through protocols across the network in wired and wireless using routing algorithms.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Outline the basic concepts of data communications including the key aspects of networking and their interrelationship, packet, circuit and cell switching as internal and external operations, physical structures, types, models, and internetworking
CO2	Make use of different types of bit errors and the concept of bit redundancy for error detection and error correction.
CO3	Identify the suitable design parameters and algorithms for assuring quality of service and internetworking in various internet protocols.
CO4	Interpret transport protocols (TCP, UDP) for measuring the network performance
CO5	Illustrate the various protocols (FTP, SMTP, TELNET, EMAIL, and WWW) and standards (DNS) in data communications among networks.
CO6	Compare various networking models (OSI, TCP/IP) in terms of design parameters and communication modes.

IV. COURSE CONTENT:

MODULE –I: INTRODUCTION (09)

Introduction: Networks, network types, internet history, standards and administration; Network models: Protocol layering, TCP/IP protocol suite, the OSI model Transmission media: Introduction, guided media, unguided media; Switching: Introduction, circuit switched networks, packet switching.

MODULE –II: DATA LINK LAYER (10)

Introduction: Link layer addressing; Error detection and correction: Cyclic codes, checksum, forward error correction; Data link control: DLC services, data link layer protocols, media access control: Random access, virtual LAN.

MODULE –III: NETWORK LAYER (10)
Network layer design issues, routing algorithms, congestion control algorithms, quality of service, and internetworking.

The network layer on the internet: IPv4 addresses, IPv6, internet control protocols, OSPF (Open Shortest Path First), IP(Internet Protocol).

MODULE –IV: TRANSPORT LAYER (10)
The transport service, elements of transport protocols, congestion control; The internet transport protocols: UDP (User Datagram Protocol), TCP (Transport Control Protocol), performance problems in computer networks, network
performance measurement.

MODULE-V: THE LINK LAYER LAYER and LANs (09)
Introduction to the link layer, services provided by the Link layer, DOCSIS-The Link layer protocol for cable internet access, Link virtualization, Multiprotocol label switching (MPLS),Data Centre networking.

V. TEXT BOOKS:

1. Behrouz A. Forouzan, "Data Communications and Networking with TCPIP Protocol Suite", Tata McGraw-Hill, 6th edition, 2022.
2. Andrew S. Tanenbaum, David.j.Wetherall, "Computer Networks", Prentice-Hall, 5th edition, 2010.

VI. REFERENCE BOOKS:

1. Douglas E. Comer, "Internetworking with TCP/IP ", Prentice-Hall, 5th edition, 2011.
2. Peterson, Davie, Elsevier, "Computer Networks", 5th edition, 2011.
3. Kurose, James F," Computer Networking": a top-down approach," : 7th edition. Hoboken, New Jersey: Pearson, 2017.

VII. ELECTRONICS RESOURCES:

1. http://computer.howstuffworks.com/computer-networking-channel.htm
2. https://www.geeksforgeeks.org/layers-osi-model/
3. https://www.wikilectures.eu/w/Computer_Network
4. https://technet.microsoft.com/en-us/network/default.aspx

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

COURSE CONTENT

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY								
IV Semester: CSE / IT / CSE (CS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSD16	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

Design and analysis of algorithm lab provides hands on experience in implementing different algorithmic paradigms and develops competence in choosing appropriate data structure to improve efficiency of technique used. This laboratory implements sorting techniques using divide and conquer strategy, shortest distance algorithms based on Greedy, Dynamic programming techniques, Minimum spanning tree construction and applications of Backtracking, Branch and Bound. This is essential for developing software in areas Information storage and retrieval, Transportation through networks, Graph theory and Optimization problems.

II. COURSE OBJECTIVES

The students will try to learn:

I. The selection of Algorithmic technique and Data structures required for efficient development of technical and engineering applications.
II. The algorithmic design paradigms and methods for identifying solutions of optimization problems.
III. Implementation of different algorithms for the similar problems to compare their performance.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Apply divide and conquer strategy to organize the data in ascending or descending order
CO2	Make use of algorithmic design paradigms to determine shortest distance and transitive closure of directed or undirected graphs
CO3	Utilize greedy technique for generating minimum cost spanning tree of a graph.
CO4	Compare the efficiencies of traversal problems using different tree and graph traversal algorithms.
CO5	Utilize backtracking method for solving puzzles involving building solutions incrementally.
CO6	Examine branch and bound approach for solving combinatorial optimization problems.

IV. COURSE CONTENT:

EXERCISES FOR DEDIGN AND ANALYSIS OF ALGORITHMS LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Implicit Recursion

A specific type of recursion called **implicit recursion** occurs when a function calls itself without making an explicit recursive call. This can occur when a function calls another function, which then calls the original code once again and starts a recursive execution of the original function.

Using implicit recursion find the second-largest elements from the array.

In this case, the **find second largest** method calls the **find_largest()** function via implicit recursion to locate the second-largest number in a provided list of numbers. Implicit recursion can be used in this way to get the second-largest integer without having to write any more code

Input: nums = [1, 2, 3, 4, 5]

Output: 4

```
class IR {

  public static int findLargest(List<Integer> numbers)
  {
     // Write code here
     …
  }

  public static int
    findSecondLargest(List<Integer> numbers)
  {
     // Write code here
     …
  }

  public static void main(String[] args)
  {
    List<Integer> numbers
      = Arrays.asList(1, 2, 3, 4, 5);

    // Function call
    int secondLargest = findSecondLargest(numbers);
    System.out.println(secondLargest);
  }
}
```

1.2 Towers of Hanoi

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

Input: 2

Output: Disk 1 moved from A to B

Disk 2 moved from A to C

Disk 1 moved from B to C

Input: 3

Output: Disk 1 moved from A to C

Disk 2 moved from A to B

Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

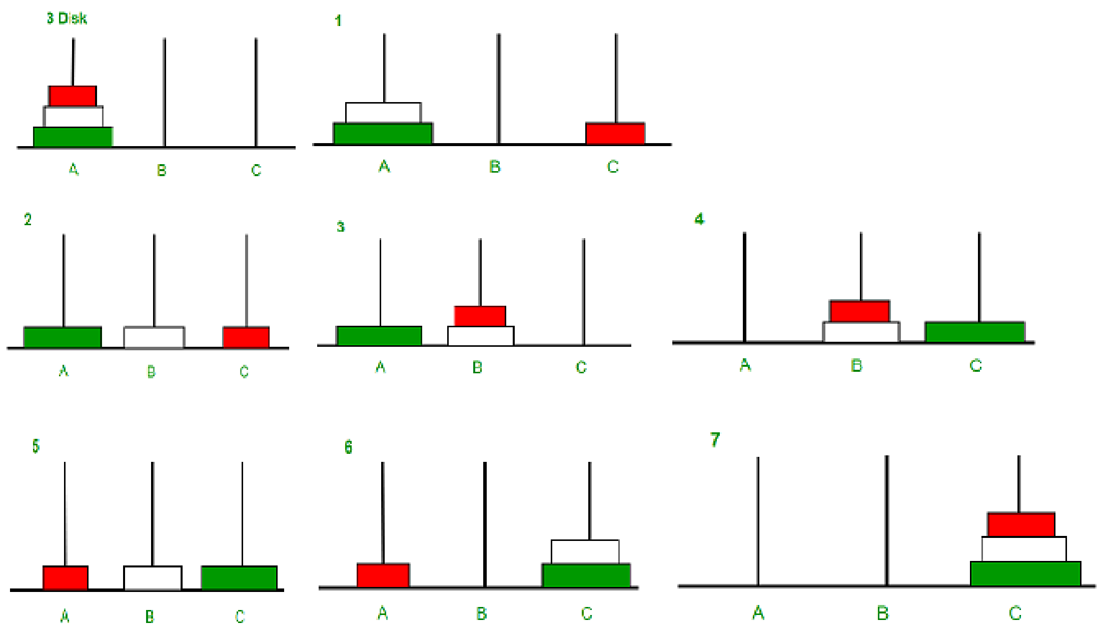
Tower of Hanoi using Recursion:

The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:

- Shift 'N-1' disks from 'A' to 'B', using C.
- Shift last disk from 'A' to 'C'.
- Shift 'N-1' disks from 'B' to 'C', using A.

Follow the steps below to solve the problem:

- Create a function towerOfHanoi where pass the N (current number of disk), from_rod, to_rod, aux_rod.
- Make a function call for N – 1 th disk.
- Then print the current the disk along with from_rod and to_rod
- Again make a function call for N – 1 th disk.



```
// Recursive java function to solve Tower of Hanoi
class TOH {
    static void towerOfHanoi(int n, char from_rod,
                             char to_rod, char aux_rod)
    {
        if (n == 0) {
            return;
        }
        // Write code here
        …
    }

    //Driver code

    public static void main(String args[])
    {
        int N = 3;

        // A, B and C are names of rods
        towerOfHanoi(N, 'A', 'C', 'B');
    }
}
```

1.3 Recursively Remove all Adjacent Duplicates

Given a string, recursively remove adjacent duplicate characters from the string. The output string should not have any adjacent duplicates.

Input: s = “azxxzy”

Output: “ay”

Explanation:

- First “azxxzy” is reduced to “azzy”.
- The string “azzy” contains duplicates
- So it is further reduced to “ay”

Input: “caaabbbaacdddd”

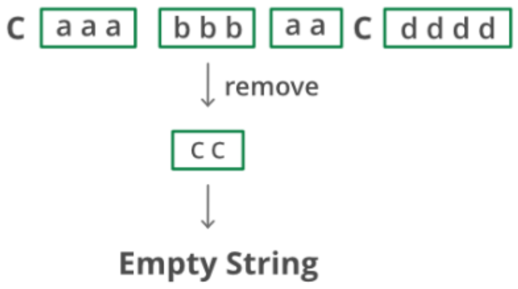
Output: Empty String

Input: “acaaabbbacdddd”

Output: “acac”

Procedure to remove duplicates:

- Start from the leftmost character and remove duplicates at left corner if there are any.
- The first character must be different from its adjacent now. Recur for string of length n-1 (string without first character).
- Let the string obtained after reducing right substring of length n-1 be rem_str. There are three possible cases
  - If first character of rem_str matches with the first character of original string, remove the first character from rem_str.
  - If remaining string becomes empty and last removed character is same as first character of original string. Return empty string.
  - Else, append the first character of the original string at the beginning of rem_str.
- Return rem_str.



```
// Java program to remove all adjacent duplicates from a string

class RD {

    static char last_removed; //will store the last char
                              // removed during recursion

    // Recursively removes adjacent duplicates from str and
    // returns new string. last_removed is a pointer to
    // last_removed character
    static String removeUtil(String str)
    {

        // If length of string is 1 or 0
        if (str.length() == 0 || str.length() == 1)
            return str;

        // Write code here
        …
    }

    static String remove(String str)
    {
        last_removed = '\0';
        return removeUtil(str);
    }

    // Driver code
    public static void main(String args[])
    {

        String str1 = "azxxxzy";
        System.out.println(remove(str1));

        String str2 = "caaabbbaac";
        System.out.println(remove(str2));

        String str3 = " gghhg ";
        System.out.println(remove(str3));

       String str4 = " aaaacddddcappp ";
        System.out.println(remove(str4));

       String str5 = " aaaaaaaaaa ";
        System.out.println(remove(str5));



    }
}
```

1.4 Randomized Algorithm - Shuffle a deck of cards

An algorithm that uses random numbers to decide what to do next anywhere in its logic is called Randomized Algorithm

Given a deck of cards, the task is to shuffle them

Output:

```
29 27 20 23 26 21 35 51 15 18 46 32 33 19

24 30 3 45 40 34 16 11 36 50 17 10 7 5 4

39 6 47 38 28 13 44 49 1 8 42 43 48 0 12

37 41 25 2 31 14 22
```

Output will be different each time because of the random function used in the program

Procedure

1. First, fill the array with the values in order.
2. Go through the array and exchange each element with the randomly chosen element in the range
    from itself to the end.
   // It is possible that an element will be swap
   // with itself, but there is no problem with that. If any of them become zero, return 0

```
import java.util.Random;

class SDC {
    // Function which shuffle and print the array
    public static void shuffle(int card[], int n)
    {

        Random rand = new Random();

        // Write code here
        …

    // Driver code
    public static void main(String[] args)
    {
        // Array from 0 to 51
        int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8,
                   9, 10, 11, 12, 13, 14, 15,
                   16, 17, 18, 19, 20, 21, 22,
                   23, 24, 25, 26, 27, 28, 29,
                   30, 31, 32, 33, 34, 35, 36,
                   37, 38, 39, 40, 41, 42, 43,
                   44, 45, 46, 47, 48, 49, 50,
                   51};

        shuffle(a, 52);

        // Printing all shuffled elements of cards
        for (int i = 0; i < 52; i ++)
           System.out.print(a[i]+" ");

    }
}
```

1.5 Strong Password Suggester using Randomized algorithm

Given a password entered by the user, check its strength and suggest some password if it is not strong.

Criteria for strong password is as follows :
A password is strong if it has :

1. At least 8 characters
2. At least one special char
3. At least one number
4. At least one upper and one lower case char.

Input : keshav123

Output : Suggested Password

k(eshav12G3

keshav1@A23

keGshav1$23

kesXhav@123

keAshav12$3

kesKhav@123

kes$hav12N3

$kesRhav123

Input: rakesh@1995kumar

Output: Your Password is Strong

Procedure:

Check the input password for its strength if it fulfills all the criteria, then it is strong password else check for the absent characters in it, and return the randomly generated password to the user.

```
// Java code to suggest strong password

public class SSP {

    // adding more characters to suggest strong password
    static StringBuilder add_more_char(
                        StringBuilder str, int need)
    {
        // Write code here
       …
    }

    // make powerful String
    static StringBuilder suggester(int l, int u, int d,
                              int s, StringBuilder str)
    {
        // Write code here
       …
    }

    /* make_password function :This function is used
    to check strongness and if input String is not
    strong, it will suggest*/
    static void generate_password(int n, StringBuilder p)
    {

       // Write code here
       …
    }

    // Driver code
    public static void main(String[] args)
    {
        StringBuilder input_String = new StringBuilder("iare@2024");
        generate_password(input_String.length(), input_String);
    }
}
```

1.6 Reservoir Sampling using Randomized Algorithm

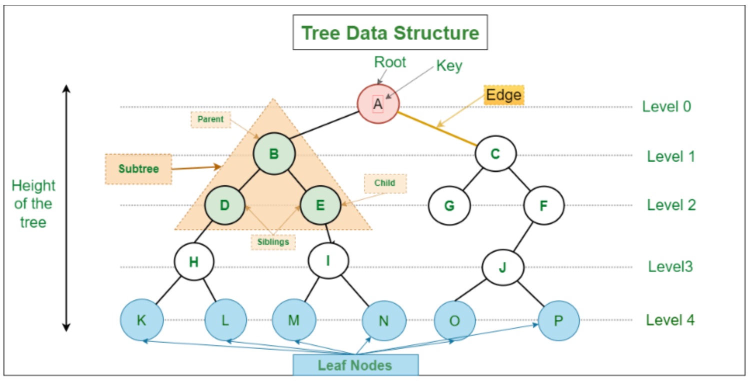
Reservoir Sampling is a family of randomized algorithms for randomly choosing *k* samples from a list of *n* items, where *n* is either a very large or unknown number. Typically, *n* is large enough that the list doesn't fit into main memory.
Example: A list of search queries in Google and Facebook.
So we are given a big array (or stream) of numbers (to simplify), and we need to write an efficient function to randomly select *k* numbers where *1 <= k <= n*.
Let the input array be *stream []*.
A **simple solution** is to create an array *reservoir []* of maximum size *k*. One by one randomly select an item from *stream [0..n-1]*. If the selected item is not previously selected, then put it in *reservoir []*. To check if an item is previously selected or not, we need to search the item in *reservoir []*. The time complexity of this algorithm will be *O(k^2)*. This can be costly if *k* is big. Also, this is not efficient if the input is in the form of a stream.
It **can be solved in O(n) time**. The solution also suits well for input in the form of stream.
Steps to get best solution:
1) Create an array *reservoir r[0..k-1]* and copy first *k* items of *stream[]* to it.
2) Now one by one consider all items from *(k+1)* th item to *n*th item
a) Generate a random number from 0 to *i* where *i* is the index of the current item in *stream[]*. Let the generated random number is *j*.
b**)** If *j* is in range 0 to *k-1*, *replace reservoir[j] with stream[i]*

```
// An efficient Java program to randomly
// select k items from a stream of items
public class ReservoirSampling {

    // A function to randomly select k items from
    // stream[0..n-1].
    static void selectKItems(int stream[], int n, int k)
    {
        // Write code here
        …

        System.out.println(
            "Following are k randomly selected items");
        System.out.println(Arrays.toString(reservoir));
    }

    // Driver Program to test above method
    public static void main(String[] args)
    {
        int stream[]
            = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
        int n = stream.length;
        int k = 5;
        selectKItems(stream, n, k);
    }
}
```

2. Trees and Graphs

2.1 Tree Creation and Traversal Techniques (Recursive)

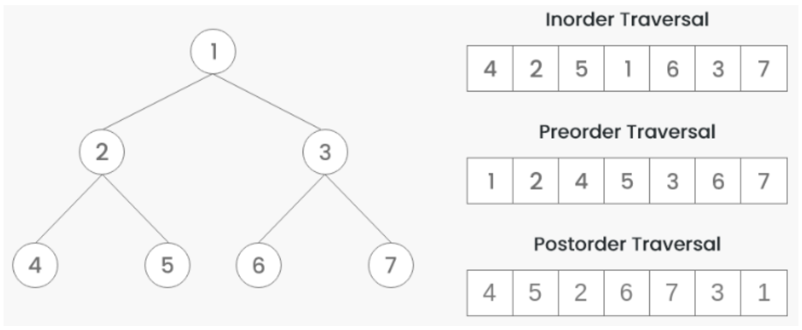
A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.



Creation of Binary Tree:

```
// Demonstration of Tree Basic Terminologies

class BT {

    // Function to print the parent of each node
    public static void
    printParents(int node, Vector<Vector<Integer> > adj,
                 int parent)
    {
        // Write code here
        …
    }

    // Function to print the children of each node
    public static void
    printChildren(int Root, Vector<Vector<Integer> > adj)
    {
        // Write code here
        …
    }

    // Function to print the leaf nodes
    public static void
    printLeafNodes(int Root, Vector<Vector<Integer> > adj)
    {
        // Write code here
        …
    }

    // Function to print the degrees of each node
    public static void
    printDegrees(int Root, Vector<Vector<Integer> > adj)
    {
        // Write code here
        …
    }

    // Driver code
    public static void main(String[] args)
    {

        // Number of nodes
        int N = 7, Root = 1;

        // Adjacency list to store the tree
        Vector<Vector<Integer> > adj
            = new Vector<Vector<Integer> >();
        for (int i = 0; i < N + 1; i++) {
            adj.add(new Vector<Integer>());
        }

        // Creating the tree
        adj.get(1).add(2);
        adj.get(2).add(1);

        adj.get(1).add(3);
        adj.get(3).add(1);

        adj.get(1).add(4);
        adj.get(4).add(1);

        adj.get(2).add(5);
        adj.get(5).add(2);

        adj.get(2).add(6);
        adj.get(6).add(2);

        adj.get(4).add(7);
        adj.get(7).add(4);

        // Printing the parents of each node
        System.out.println("The parents of each node are:");
        printParents(Root, adj, 0);

        // Printing the children of each node
        System.out.println(
            "The children of each node are:");
        printChildren(Root, adj);

        // Printing the leaf nodes in the tree
        System.out.println(
            "The leaf nodes of the tree are:");
        printLeafNodes(Root, adj);

        // Printing the degrees of each node
        System.out.println("The degrees of each node are:");
        printDegrees(Root, adj);
    }
}
```

A binary tree data structure can be traversed in following ways:
1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Level Order Traversal



Algorithm Inorder (tree)

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Algorithm Preorder (tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left->subtree)
3. Traverse the right subtree, i.e., call Preorder(right->subtree)

Algorithm Postorder (tree)

1. Traverse the left subtree, i.e., call Postorder(left->subtree)
2. Traverse the right subtree, i.e., call Postorder(right->subtree)
3. Visit the root.

```
// Java program for different tree traversals

// Class containing left and right child of current
// node and key value
class Node {
    int key;
    Node left, right;
    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree {

    // Root of Binary Tree
```

```java
    Node root;

    BinaryTree() { root = null; }

    // Given a binary tree, print its nodes in inorder
    void printInorder(Node node)
    {
        // Write code here
        …
    }
    // Given a binary tree, print its nodes in preorder
    void printPreorder(Node node)
    {
        // Write code here
        …
    }
    void printPostorder(Node node)
    {
        // Write code here
        …    }


    // Driver code
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        // Function call
        System.out.println(
            "Inorder traversal of binary tree is ");
        tree.printInorder(tree.root);
        System.out.println(
            "Preorder traversal of binary tree is ");
        tree.printPreorder(tree.root);
        System.out.println(
            "Postorder traversal of binary tree is ");
        tree.printPostorder(tree.root);


    }
}
```

2.2 Tree Creation and Traversal Techniques (Iterative)

Inorder Traversal using Stack:
As we already know, recursion can also be implemented using stack. Here also we can use a stack to perform inorder traversal of a Binary Tree. Below is the algorithm for traversing a binary tree using stack.

1. Create an empty stack (say S).

2. Initialize the current node as root.

3. Push the current node to S and set current = current->left until current is NULL

4. If current is NULL and the stack is not empty then:

   - Pop the top item from the stack.

   - Print the popped item and set current = popped_item->right

   - Go to step 3.

5. If current is NULL and the stack is empty then we are done.

preorder Traversal using Stack:
Following is a simple stack based iterative process to print Preorder traversal.

1. Create an empty stack nodeStack and push root node to stack.

2. Do the following while nodeStack is not empty.

   1. Pop an item from the stack and print it.

   2. Push right child of a popped item to stack

   3. Push left child of a popped item to stack

The right child is pushed before the left child to make sure that the left subtree is processed first.

postorder Traversal using Stack:

```
1.1 Create an empty stack
2.1 Do following while root is not NULL
   a) Push root's right child and then root to stack.
   b) Set root as root's left child.
2.2 Pop an item from stack and set it as root.
   a) If the popped item has a right child and the right child
      is at top of stack, then remove the right child from stack,
      push the root back and set root as root's right child.
   b) Else print root's data and set root as NULL.
2.3 Repeat steps 2.1 and 2.2 while stack is not empty.
```

```
//Inorder Traversal
class Node
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}
```

```
// Class to print the inorder traversal
class BinaryTree
{
    Node root;
    void inorder()
    {
        if (root == null)
            return;
        Stack<Node> s = new Stack<Node>();
        Node curr = root;
        // Traverse the tree
        // Write code here
            ……
    }
     public static void main(String args[])
    {
        // Creating a binary tree and entering
        // the nodes
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.inorder();
    }
}
// Preorder traversal
// A binary tree node
class Node {
    int data;
    Node left, right;
    Node(int item)
    {
        data = item;
        left = right = null;
    }
}
class BinaryTree {
    Node root;
    void iterativePreorder()    {
        iterativePreorder(root);
    }

    // An iterative process to print preorder traversal of Binary tree
    void iterativePreorder(Node node)
    {
        // Base Case
        if (node == null) {
            return;
        }
        // Write code here
            ……
    }

    // driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(10);
        tree.root.left = new Node(8);
        tree.root.right = new Node(2);
        tree.root.left.left = new Node(3);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(2);
        tree.iterativePreorder();
    }
}
```

```
//Postorder Traversal
// A binary tree node
class Node {
    int data;
    Node left, right;
    Node(int item)
    {
        data = item;
        left = right;
    }
}
class BinaryTree {
    Node root;
    ArrayList<Integer> list = new ArrayList<Integer>();
     // Write code here
       ………
    }
     // Driver program to test above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();

        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        tree.root.right.left = new Node(6);
        tree.root.right.right = new Node(7);

        ArrayList<Integer> mylist = tree.postOrderIterative(tree.root);

        System.out.println(
            "Post order traversal of binary tree is :");
        System.out.println(mylist);
    }
}
```

2.3 Hashing

Hashing is a technique or process of mapping keys, and values into the hash table by using a hash function. It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.In this case, the **find_second_largest** method calls the **find_largest()** function via implicit recursion to locate the second-largest number in a provided list of numbers. Implicit recursion can be used in this way to get the second-largest integer without having to write any more code

Let a hash function H(x) maps the value **x** at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.

Given four sorted arrays each of size **n** of distinct elements. Given a value **x**. The problem is to count all **quadruples**(group of four numbers) from all the four arrays whose sum is equal to **x**.
Note: The quadruple has an element from each of the four arrays.
Examples:
Input : arr1 = {1, 4, 5, 6},
arr2 = {2, 3, 7, 8},
arr3 = {1, 4, 6, 10},
arr4 = {2, 4, 7, 8}
n = 4, x = 30
Output : 4
The quadruples are:
(4, 8, 10, 8), (5, 7, 10, 8),
(5, 8, 10, 7), (6, 7, 10, 7)
Input : For the same above given fours arrays
x = 25
Output : 14

Procedure using Hashing
Create a hash table where **(key, value)** tuples are represented as **(sum, frequency)** tuples. Here the sum are obtained from the pairs of 1st and 2nd array and their frequency count is maintained in the hash table. Now, generate all pairs from the 3rd and 4th array. For each pair so generated, find the sum

of elements in the pair. Let it be **p_sum**. For each **p_sum**, check whether **(x – p_sum)** exists in the hash table or not. If it exists, then add the frequency of **(x – p_sum)** to the **count** of quadruples

```
// Java implementation to count quadruples from four sorted arrays
// whose sum is equal to a given value x
class HT {

    static int countQuadruples(int arr1[], int arr2[],
            int arr3[], int arr4[], int n, int x) {
        int count = 0;
        // Write code here
       ………
    }

// Driver program to test above
    public static void main(String[] args) {

        // four sorted arrays each of size 'n'
        int arr1[] = {1, 4, 5, 6};
        int arr2[] = {2, 3, 7, 8};
        int arr3[] = {1, 4, 6, 10};
        int arr4[] = {2, 4, 7, 8};

        int n = arr1.length;
        int x = 30;
        System.out.println("Count = "
                + countQuadruples(arr1, arr2, arr3,
                        arr4, n, x));

    }
}
```

2.4 Graph Traversal Techniques

Graph Traversal Techniques:

a. **Breadth First Traversal (BFT)**

b. **Depth First Traversal (BFT)**

The **Breadth First Traversal (BFT)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
For a given graph G, print BFS traversal from a given source vertex.

```
class Graph {
    int vertices;
    LinkedList<Integer>[] adjList;
    @SuppressWarnings("unchecked") Graph(int vertices)
    {
        this.vertices = vertices;
        adjList = new LinkedList[vertices];
        for (int i = 0; i < vertices; ++i)
            adjList[i] = new LinkedList<>();
    }

    // Function to add an edge to the graph
    void addEdge(int u, int v) { adjList[u].add(v); }

    // Function to perform Breadth First Search on a graph
    // represented using adjacency list
    void bfs(int startNode)
    {
        // Write code here
        ………
    }
}

public class Main {
    public static void main(String[] args)
    {
        // Number of vertices in the graph
        int vertices = 5;

        // Create a graph
        Graph graph = new Graph(vertices);

        // Add edges to the graph
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 3);
        graph.addEdge(1, 4);
        graph.addEdge(2, 4);

        // Perform BFS traversal starting from vertex 0
        System.out.print(
            "Breadth First Traversal starting from vertex 0: ");
        graph.bfs(0);
    }
}


# BFT traversal from a given source vertex.

from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
            self.graph[u].append(v)

        # Function to print a BFS of graph
        def BFT(self, s):
           # Write code here
           …
 # Create a graph given in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal" " (starting from vertex 2)")
g.BFT(2)
```

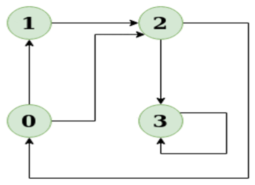
Output: Breadth First Traversal starting from vertex 0: 0 1 2 3 4

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

Depth First Traversal (DFT) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.
Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3
Output: DFS from vertex 1: 1 2 0 3

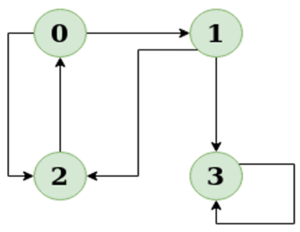
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:



```
// Java program to print DFS traversal
// from a given graph
class Graph {
    private int V;

    // Array of lists for Adjacency List Representation
    private LinkedList<Integer> adj[];

    // Constructor
    @SuppressWarnings("unchecked") Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    // Function to add an edge into the graph
    void addEdge(int v, int w)
    {
        // Add w to v's list.
        adj[v].add(w);
    }

    // A function used by DFS
    void DFSUtil(int v, boolean visited[])
    {
        // Write code here
        ………
    }

    // The function to do DFS traversal. It uses recursive DFSUtil()
    void DFS(int v)
    {
        // Write code here
        ………
    }

    // Driver Code
    public static void main(String args[])
    {
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);
        System.out.println(
            "Following is Depth First Traversal "
            + "(starting from vertex 2)");

        // Function call
        g.DFS(2);
    }
}
```

2.5 Heap Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Heap Sort Procedure:
First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
  - Swap the root element of the heap (which is the largest element) with the last element of the heap.
  - Remove the last element of the heap (which is now in the correct position).
  - Heapify the remaining elements of the heap.
- The sorted array is obtained by reversing the order of the elements in the input array.

Input: arr = [12, 11, 13, 5, 6, 7]
Output: Sorted array is 5 6 7 11 12 13

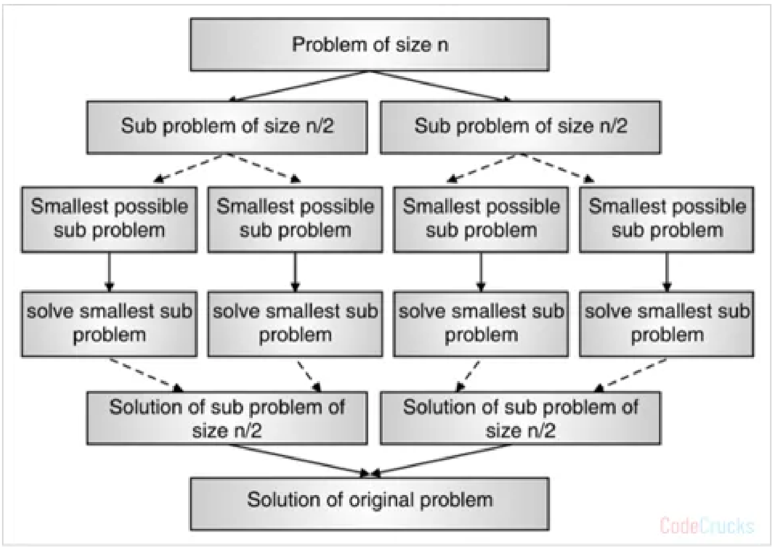
```
// Java program for implementation of Heap Sort

public class HeapSort {
    public void sort(int arr[])
    {
        int N = arr.length;

        // Build heap (rearrange array)
        for (int i = N / 2 - 1; i >= 0; i--)
            heapify(arr, N, i);

        // One by one extract an element from heap
        for (int i = N - 1; i > 0; i--) {
            // Move current root to end
            // Write code here
              .........

            // call max heapify on the reduced heap
            heapify(arr, i, 0);
        }
    }

    // To heapify a subtree rooted with node i which is
    // an index in arr[]. n is size of heap
    void heapify(int arr[], int N, int i)
    {
        // Write code here
        ………    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        // Write code here
        ………
    }

    // Driver's code
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int N = arr.length;

        // Function call
        HeapSort ob = new HeapSort();
        ob.sort(arr);

        System.out.println("Sorted array is");
        printArray(arr);
    }
}
```

3. Divide and Conquer

General strategy of Divide and Conquer:
Divide and conquer algorithm operates in three stages:

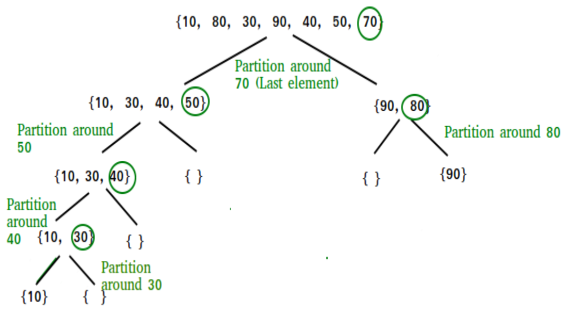
- **Divide:** Divide the problem recursively into smaller subproblems.
- **Solve:** Subproblems are solved independently.
- **Combine:** Combine subproblem solutions in order to deduce the answer to the original large problem.

Because subproblems are identical to the main problem but have smaller parameters, they can be readily solved using recursion. When a subproblem is reduced to its lowest feasible size, it is solved, and the results are recursively integrated to produce a solution to the original larger problem.
Divide and conquer is a top-down, multi-branched recursive method. Each branch represents a subproblem and calls itself with a smaller argument. Understanding and developing divide and conquer algorithms requires expertise and sound reasoning. Divide and Conquer approach depicted graphically in the following figure. Sub Problems need not be exactly n/2 in size. Size of partition and number of partitions depends on application problem.


Problem of size n
Sub problem of size n/2
Sub problem of size n/2
Smallest possible sub problem
Smallest possible sub problem
Smallest possible sub problem
Smallest possible sub problem
solve smallest sub problem
solve smallest sub problem
solve smallest sub problem
solve smallest sub problem
Solution of sub problem of size n/2
Solution of sub problem of size n/2
Solution of original problem
CodeCrucks

3.1 Quick Sort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



The quick sort method can be summarized in three steps:

1. **Pick:** Select a pivot element.
2. **Divide:** Split the problem set, move smaller parts to the left of the pivot and larger items to the right.
3. **Repeat and combine:** Repeat the steps and combine the arrays that have previously been sorted.

Algorithm for Quick Sort Function:

```
//start –> Starting index,  end --> Ending index
Quicksort(array, start, end)
{
        if (start < end)
        {
                pIndex = Partition(A, start, end)
                Quicksort(A,start,pIndex-1)
                Quicksort(A,pIndex+1, end)
        }
}
```

Algorithm for Partition Function:

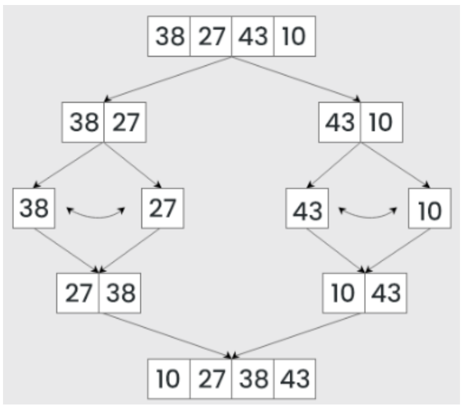
```
partition (array, start, end)
{
        // Setting rightmost Index as pivot
        pivot = arr[end];

        i = (start - 1)  // Index of smaller element and indicates the
            // right position of pivot found so far
        for (j = start; j <= end- 1; j++)
        {
                // If current element is smaller than the pivot
                if (arr[j] < pivot)
                {
                        i++;    // increment index of smaller element
                        swap arr[i] and arr[j]
                }
        }
        swap arr[i + 1] and arr[end])
        return (i + 1)
}
```

Input: arr = [10, 80, 30, 90, 40, 50, 70]
Output: arr = [10, 30, 40, 50, 70, 80, 90]

```
class QS {

    // A utility function to swap two elements
    static void swap(int[] arr, int i, int j)
    {
        // Write code here
         …
    }

    static int partition(int[] arr, int low, int high)
    {
        // Choosing the pivot
        int pivot = arr[high];

        // Write code here
         …
        swap(arr, i + 1, high);
        return (i + 1);
    }

    static void quickSort(int[] arr, int low, int high)
    {
        // Write code here
         …
    }
    // To print sorted array
    public static void printArr(int[] arr)
    {
        // Write code here
         …
    }

    // Driver Code
    public static void main(String[] args)
    {
        int[] arr = { 10, 7, 8, 9, 1, 5 };
        int N = arr.length;

        // Function call
        quickSort(arr, 0, N - 1);
        System.out.println("Sorted array:");
        printArr(arr);
    }
}
```

3.2 Merge Sort

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



Input: arr = [12, 11, 13, 5, 6, 7]
Output: arr = [5, 6, 7, 11, 12, 13]

```
class MergeSort {
    void merge(int arr[], int l, int m, int r)
    {
        // Write code here
         …    }

    void sort(int arr[], int l, int r)
    {
        if (l < r) {
             // Find the middle point
            int m = l + (r - l) / 2;
            // Sort first and second halves
            sort(arr, l, m);
            sort(arr, m + 1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }

    // A utility function to print array of size n
    static void printArray(int arr[])
    {
        // Write code here
         …
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = { 12, 11, 13, 5, 6, 7 };

        System.out.println("Given array is");
        printArray(arr);

        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.length - 1);

        System.out.println("\nSorted array is");
        printArray(arr);
    }
}
```

3.3 Karatsuba Algorithm for fastest multiplication

Given two binary strings that represent value of two integers, find the product of two strings using **Karatsuba algorithm for fast multiplication**. For example, if the first bit string is "1100" and second bit string is "1010", output should be 120. For simplicity, let the length of two strings be same and be n.

or this algorithm, two n-digit numbers are taken as the input and the product of the two number is obtained as the output.

Step 1 – In this algorithm we assume that n is a power of 2.

Step 2 – If n = 1 then we use multiplication tables to find P = XY.

Step 3 – If n > 1, the n-digit numbers are split in half and represent the number using the formulae –

$$X = 10n/2X1 + X2$$

$$Y = 10n/2Y1 + Y2$$

where, X_1, X_2, Y_1, Y_2 each have n/2 digits.

Step 4 – Take a variable Z = W – (U + V), where,

U = X1Y1, V = X2Y2

W = (X1 + X2) (Y1 + Y2), Z = X1Y2 + X2Y1.

Step 5 – Then, the product P is obtained after substituting the values in the formula –

P = 10n(U) + 10n/2(Z) + V

P = 10n (X1Y1) + 10n/2 (X1Y2 + X2Y1) + X2Y2.

Step 6 – Recursively call the algorithm by passing the sub problems (X_1, Y_1), (X_2, Y_2) and (X_1 + X_2, Y_1 + Y_2) separately. Store the returned values in variables U, V and W respectively.

```
public class Main {
   static long karatsuba(long X, long Y) {
      // Base Case
      if (X < 10 && Y < 10)
         return X * Y;
      // Write Code Here
         ……

   }
   static int get_size(long value) {
      // Write Code Here
         ……
   }
   public static void main(String args[]) {
      // two numbers
      long x = 145623;
      long y = 653324;
      System.out.print("The final product is: ");
      long product = karatsuba(x, y);
      System.out.println(product);
   }
}
```

3.4 Strassen's Matrix Multiplication

Strassen's Matrix Multiplication is the divide and conquer approach to solve the matrix multiplication problems. The usual matrix multiplication method multiplies each row with each column to achieve the product matrix. The time complexity taken by this approach is **$O(n^3)$**, since it takes two loops to multiply. Strassen's method was introduced to reduce the time complexity from **$O(n^3)$** to **$O(n^{\log 7})$**

Strassen's Matrix multiplication can be performed only on square matrices where n is a power of 2. Order of both of the matrices are n × n.

Divide X, Y and Z into four (n/2)×(n/2) matrices as represented below –

$$Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix} \qquad X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Using Strassen's Algorithm compute the following −

$$M_1 := (A + C) \times (E + F)$$
$$M_2 := (B + D) \times (G + H)$$
$$M_3 := (A - D) \times (E + H)$$
$$M_4 := A \times (F - H)$$
$$M_5 := (C + D) \times (E)$$
$$M_6 := (A + B) \times (H)$$
$$M_7 := D \times (G - E)$$

Then,

$$I := M_2 + M_3 - M_6 - M_7$$
$$J := M_4 + M_6$$
$$K := M_5 + M_7$$
$$L := M_1 - M_3 - M_4 - M_5$$

```
/** Java Program to Implement Strassen Algorithm**/
import java.util.Scanner;
/** Class Strassen **/
public class Strassen
{      /** Function to multiply matrices **/
       public int[][] multiply(int[][] A, int[][] B)
       {
              int n = A.length;
              int[][] R = new int[n][n];
              /** base case **/
              if (n == 1)
                     R[0][0] = A[0][0] * B[0][0];
              else
              {   ………….Write Code Here………………

              }
       }
       /** Function to sub two matrices **/
       public int[][] sub(int[][] A, int[][] B)
       {
              ……….Write Code Here……….
              return C;
       }
       /** Function to add two matrices **/
       public int[][] add(int[][] A, int[][] B)
       {
              ……Write Code Here………….
              return C;
       }
       /** Function to split parent matrix into child matrices **/
       public void split(int[][] P, int[][] C, int iB, int jB) { ….Write
Code….}
       /** Function to join child matrices intp parent matrix **/
       public void join(int[][] C, int[][] P, int iB, int jB) { ….Write
Code….}
       /** Main function **/
       public static void main (String[] args)
```

```
        {
                ….Write Code Here…..
        }
}
```

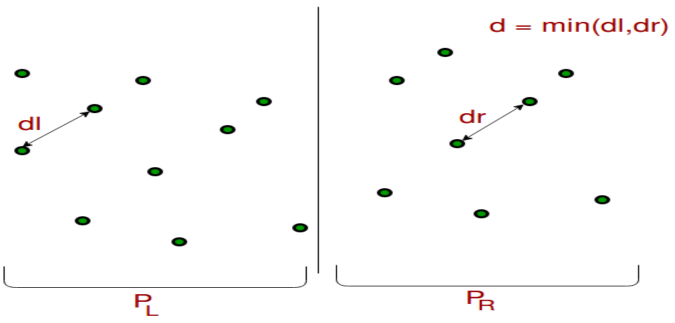
3.5 Closest Pair of Points

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.

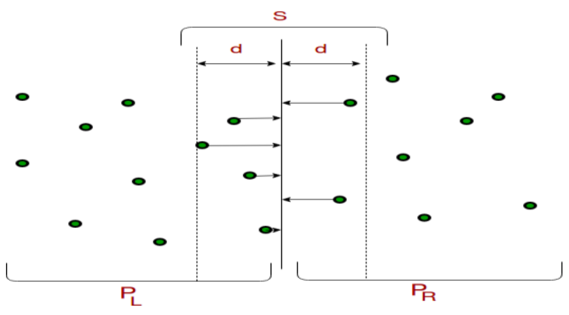
$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. We can calculate the smallest distance in O(nLogn) time using Divide and Conquer strategy. In this post, a O(n x (Logn)^2) approach is discussed. We will be discussing a O(nLogn) approach in a separate post.

Algorithm
Following are the detailed steps of a O (n (Logn)^2) algorithm.
Input: An array of n points P []
Output: The smallest distance between two points in the given array.
As a pre-processing step, the input array is sorted according to x coordinates.
1) Find the middle point in the sorted array, we can take P[n/2] as middle point.
2) Divide the given array in two halves. The first subarray contains points from P[0] to P[n/2]. The second subarray contains points from P[n/2+1] to P[n-1].
3) Recursively find the smallest distances in both subarrays. Let the distances be dl and dr. Find the minimum of dl and dr. Let the minimum be d.



4) From the above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from the left half and the other is from the right half. Consider the vertical line passing through P[n/2] and find all points whose x coordinate is closer than d to the middle vertical line. Build an array strip [] of all such points.



5) Sort the array strip[] according to y coordinates. This step is O(nLogn). It can be optimized to O(n) by recursively sorting and merging.
6) Find the smallest distance in strip[]. This is tricky. From the first look, it seems to be a O(n^2) step, but it is actually O(n). It can be proved geometrically that for every point in the strip, we only need to check at most 7 points after it (note that strip is sorted according to Y coordinate).
7) Finally return the minimum of d and distance calculated in the above step (step 6)

```
import java.text.DecimalFormat;
import java.util.Arrays;
import java.util.Comparator;
// A divide and conquer program in Java to find the smallest distance from
// a given set of points A structure to represent a Point in 2D plane
class Point {
public int x;
public int y;
Point(int x, int y) {
      this.x = x;
      this.y = y;
}

// A utility function to find the distance between two points
public static float dist(Point p1, Point p2) {
      return (float) Math.sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                                    (p1.y - p2.y) * (p1.y - p2.y)
                                    );
}
// A recursive function to find the smallest distance. The array P
// contains all points sorted according to x coordinate
public static float closestUtil(Point[] P, int startIndex, int endIndex)
{     // If there are 2 or 3 points, then use brute force
      if ((endIndex - startIndex) <= 3) {
      return bruteForce(P, endIndex);
      }
      ………………..Write Code Here……

public class ClosestPoint {
// Driver code
public static void main(String[] args) { ………….. Write Code Here……..
      DecimalFormat df = new DecimalFormat("#.######");
      System.out.println("The smallest distance is " +
                              df.format(Point.closest(P, P.length)));
}

}
```

4. Divide and Conquer

4.1 Tiling Problem using Divide and Conquer Algorithm

Given a n by n board where n is of form 2^k where k >= 1 (Basically n is a power of 2 with minimum value as 2). The board has one missing cell (of size 1 x 1). Fill the board using L shaped tiles. A L shaped tile is a 2 x 2 square with one cell of size 1×1 missing.

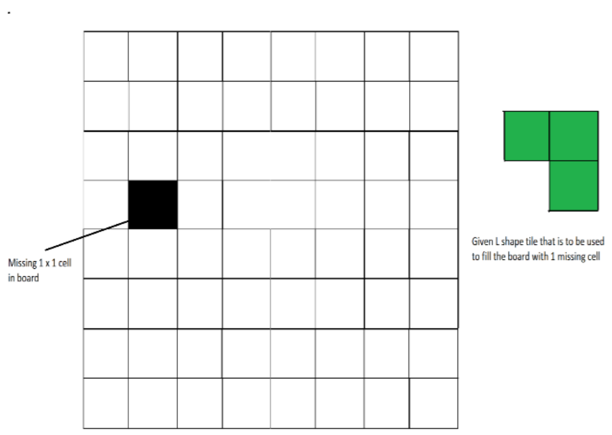

Missing 1 x 1 cell in board
Given L shape tile that is to be used to fill the board with 1 missing cell

Figure 1: An example input

This problem can be solved using Divide and Conquer. Below is the recursive algorithm.
// n is size of given square, p is location of missing cell
Tile(int n, Point p)

1) Base case: n = 2, A 2 x 2 square with one cell missing is nothing but a tile and can be filled with a single tile.

2) Place a L shaped tile at the center such that it does not cover the n/2 * n/2 subsquare that has a missing square. **Now all four subsquares of size n/2 x n/2 have a missing cell** (a cell that doesn't need to be filled). See figure 2 below.

3) Solve the problem recursively for following four. Let p1, p2, p3 and p4 be positions of the 4 missing cells in 4 squares.
   a) Tile(n/2, p1)
   b) Tile(n/2, p2)
   c) Tile(n/2, p3)
   d) Tile(n/2, p3)

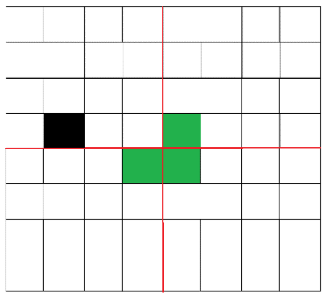


Figure 2: After placing the first tile

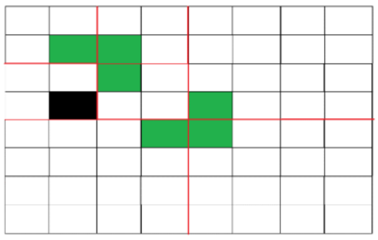


Figure 3: Recurring for the first subsquare.

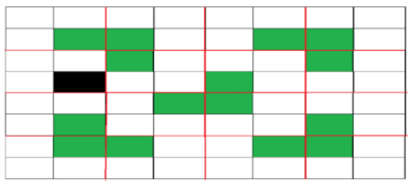


Figure 4: Shows the first step in all four subsquares.

Input : size = 2 and mark coordinates = (0, 0)
Output :

```
-1      1
1       1
```

Coordinate (0, 0) is marked. So, no tile is there. In the remaining three positions, a tile is placed with its number as 1.
Input : size = 4 and mark coordinates = (0, 0)
Output :

```
-1      3       2       2
3       3       1       2
4       1       1       5
4       4       5       5
```

```
public class TP
{
  static int size_of_grid, b, a, cnt = 0;
  static int[][] arr = new int[128][128];

  // Placing tile at the given coordinates
  static void place(int x1, int y1, int x2,
                    int y2, int x3, int y3)
  {
    // Write code here
```

```
        ……
  }
  // Quadrant names
  // 1   2
  // 3   4
  // Function based on divide and conquer
  static int tile(int n, int x, int y)
  {
    // Write code here
        ……
   }

  // Driver code
  public static void main(String[] args)
  {
    // size of box
    size_of_grid = 8;
    // Coordinates which will be marked
    a = 0; b = 0;
    // Here tile can not be placed
    arr[a][b] = -1;
    tile(size_of_grid, 0, 0);
    // The grid is
    for (int i = 0; i < size_of_grid; i++)
    {
      for (int j = 0; j < size_of_grid; j++)
        System.out.print(arr[i][j] + " ");
      System.out.println();;
    }
  }
}
```

4.2 The Skyline Problem

Given n rectangular buildings in a 2-dimensional city, computes the skyline of these buildings, eliminating hidden lines. The main task is to view buildings from a side and remove all sections that are not visible. All buildings share a common bottom and every **building** is represented by a triplet (left, ht, right)

- 'left': is the x coordinate of the left side (or wall).
- 'right': is x coordinate of right side
- 'ht': is the height of the building.

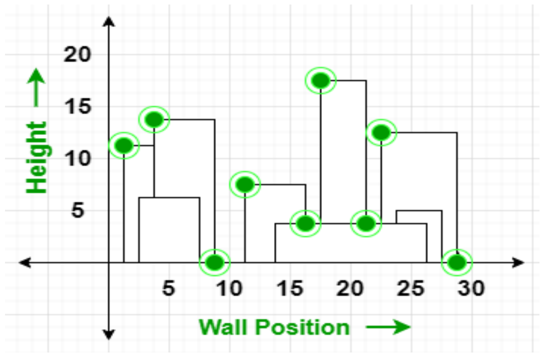
A **skyline** is a collection of rectangular strips. A rectangular **strip** is represented as a pair (left, ht) where left is x coordinate of the left side of the strip and ht is the height of the strip.

Input: buildings[][] =

{ {1, 11, 5}, {2, 6, 7}, {3, 13, 9}, {12, 7, 16}, {14, 3, 25}, {19, 18, 22}, {23, 13, 29}, {24, 4, 28} }

Output: { {1, 11}, {3, 13}, {9, 0}, {12, 7}, {16, 3}, {19, 18}, {22, 3}, {23, 13}, {29, 0} }

Explanation:
The skyline is formed based on the key-points (representing by “green” dots) eliminating hidden walls of the buildings.
The Skyline Problem:
Input: buildings[ ][ ] = { {1, 11, 5} }
Output: { {1, 11}, {5, 0}



We can find Skyline using **Divide and Conquer**. The idea is similar to Merge Sort, divide the given set of buildings in two subsets. Recursively construct skyline for two halves and finally merge the two skylines. Start from first strips of two skylines, compare **x** coordinates. Pick the strip with smaller **x** coordinate and add it to result. The height of added strip is considered as maximum of current heights from skyline1 and skyline2.

```
// A class for building
class Building {
    int left, ht, right;

    public Building(int left, int ht, int right) {
        this.left = left;
        this.ht = ht;
        this.right = right;
    }
}
// A strip in skyline
class Strip {
    int left, ht;

    public Strip(int left, int ht) {
        this.left = left;
        this.ht = ht;
    }
}
// Skyline: To represent Output(An array of strips)
class SkyLine {
    List<Strip> arr;
    int capacity, n;

    public SkyLine(int cap) {
        this.arr = new ArrayList<>();
        this.capacity = cap;
        this.n = 0;
    }

    public int count() {
        return this.n;
    }

// A function to merge another skyline to this skyline
    public SkyLine merge(SkyLine other) {
        // Write code here
            ......
    }

// Function to add a strip 'st' to array
    public void append(Strip st) {
    // Write code here
            ......
    }

// A utility function to print all strips of skyline
    public void printSkyline() {
        // Write code here
            ......
    }
}
// This function returns skyline for a given array of buildings arr[l..h].
// This function is similar to mergeSort().
class SkylineProblem {
    public static SkyLine findSkyline(Building[] arr, int l, int h) {
           // Write code here
            ......
        return res;
    }
  // Driver Code
public static void main(String[] args) {
Building[] arr = {new Building(1, 11, 5), new Building(2, 6, 7), new Building(3, 13, 9),
new Building(12, 7, 16), new Building(14, 3, 25), new Building(19, 18, 22),
new Building(23, 13, 29), new Building(24, 4, 28)};

// Find skyline for given buildings and print the skyline
SkyLine res = findSkyline(arr, 0, arr.length-1);
res.printSkyline();
}
}
```

4.3 Allocate Minimum Number of Pages from N books to M students

Given that there are **N books** and **M students**. Also given are the **number of pages in each book in ascending order**. The task is to assign books in such a way that the **maximum number of pages**

assigned to a student is minimum, with the condition that every student is assigned to read some consecutive books. Print that minimum number of pages.

Example :
Input: N = 4, pages[] = {12, 34, 67, 90}, M = 2
Output: 113
Explanation: There are 2 number of students. Books can be distributed in following combinations:

1. [12] and [34, 67, 90] -> Max number of pages is allocated to student '2' with 34 + 67 + 90 = 191 pages

2. [12, 34] and [67, 90] -> Max number of pages is allocated to student '2' with 67 + 90 = 157 pages

3. [12, 34, 67] and [90] -> Max number of pages is allocated to student '1' with 12 + 34 + 67 = 113 pages

Of the 3 cases, Option 3 has the minimum pages = 113.

Approach to solve this problem is to use Divide and Conquer, based on Binary Search idea:
Case 1: When no valid answer exists.

- If the number of students is greater than the number of books **(i.e, M > N)**, In this case at least 1 student will be left to which no book has been assigned.

Case 2: When a valid answer exists.

- The **maximum possible answer** could be when there is only one student. So, all the book will be assigned to him and the result would be the sum of pages of all the books.

- The **minimum possible answer** could be when number of student is equal to the number of book (i.e, M == N) , In this case all the students will get at most one book. So, the result would be the maximum number of pages among them **(i.e, maximum(pages[])).**

- Hence, we can apply binary search in this given range and each time we can consider the mid value as the maximum limit of pages one can get. And check for the limit if answer is valid then update the limit accordingly.

Below is the approach to solve this problem using Divide and Conquer:

- Calculate the mid and check if **mid** number of pages can be assigned to students such that all students will get at least one book.

- If yes, then update the result and check for the previous search space (end = mid-1)

- Otherwise, check for the next search space (start = mid+1)

```
public class NM {
  // Utility method to check if current minimum value is feasible or not.
  static boolean isPossible(int arr[], int n, int m,  int curr_min)
  {
    int studentsRequired = 1;
    int curr_sum = 0;
```

```
        // Write code here
          ......

        return studentsRequired <= m;
    }
    // method to find minimum pages
    static int findPages(int arr[], int n, int m)
    {
        int sum = 0;
        // return -1 if no. of books is less than no. of students
        if (n < m)
            return -1;
        int mx = arr[0];
        // Write code here
          ......
        return result;
    }
     // Driver Method
    public static void main(String[] args)
    {

        int arr[] = { 12, 34, 67,  90 };
  // Number of pages in books
        int m = 2; // No. of students
        System.out.println("Minimum number of pages = " + findPages(arr, arr.length, m));
    }
}
```

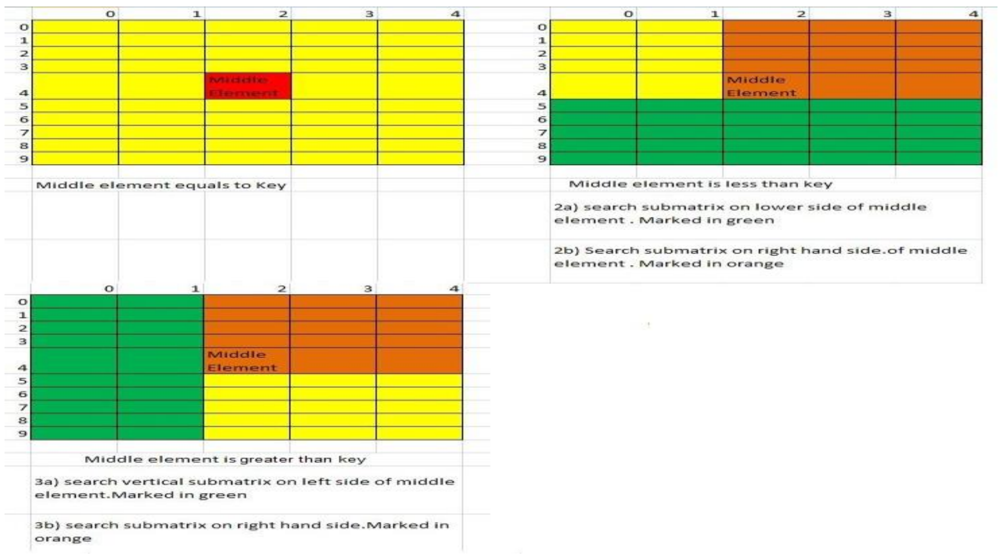
4.4 Efficiently merge `k` sorted linked lists Minimum Number of Pages from N books to M students

Given k sorted linked lists, merge them into a single list in increasing order.
We already know that Two Linked Lists can be Merged in O(n) time and O(1) space (For arrays, O(n) space is required). The idea is to pair up k lists and merge each pair in linear time using the O(1) space. After the first cycle, K/2 lists are left each of size 2×N. After the second cycle, K/4 lists are left each of size 4×N and so on. Repeat the procedure until we have only one list left.

4.5 Search in a Row-wise and Column-wise Sorted 2D Array using Divide and Conquer algorithm

Given an n x n matrix, where every row and column is sorted in increasing order. Given a key, how to decide whether this key is in the matrix. This problem will be a very good example for divide and conquer algorithm.
1) Find the middle element.
2) If middle element is same as key return.
3) If middle element is lesser than key then
....3a) search submatrix on lower side of middle element
....3b) Search submatrix on right hand side.of middle element

4) If middle element is greater than key then
....4a) search vertical submatrix on left side of middle element
....4b) search submatrix on right hand side.**Shell Sort Procedure:**


Middle Element
Middle element equals to Key
Middle element is less than key
2a) search submatrix on lower side of middle element . Marked in green
2b) Search submatrix on right hand side.of middle element . Marked in orange
Middle element is greater than key
3a) search vertical submatrix on left side of middle element.Marked in green
3b) search submatrix on right hand side.Marked in orange

```
class SearchInMatrix
{
    public static void main(String[] args)
    {
        int[][] mat = new int[][] { {10, 20, 30, 40},
                                    {15, 25, 35, 45},
                                    {27, 29, 37, 48},
                                    {32, 33, 39, 50}};
        int rowcount = 4,colCount=4,key=50;
        for (int i=0; i<rowcount; i++)
          for (int j=0; j<colCount; j++)
             search(mat, 0, rowcount-1, 0, colCount-1, mat[i][j]);
    }

    public static void search(int[][] mat, int fromRow, int toRow,
                              int fromCol, int toCol, int key)
    {
        // Write code here
           ......

    }
}
```

5. Greedy Technique

Greedy Algorithm is optimization method. When the problem has many feasible solutions with different cost or benefit, finding the best solution is known as an optimization problem and the best solution is known as the optimal solution.

The greedy algorithm derives the solution step by step, by looking at the information available at the current moment. It does not look at future prospects. Decisions are completely locally optimal. This method constructs the solution simply by looking at current benefit without exploring future possibilities and hence they are known as greedy. The choice made under greedy solution procedure are irrevocable, means once we have selected the local best solution, it cannot be backtracked. Thus, a choice made at each step in the greedy method should be:

- Feasible: choice should satisfy problem constraints.
- Locally optimal: Best solution from all feasible solution at the current stage should be selected.
- Irrevocable: Once the choice is made, it cannot be altered, i.e. if a feasible solution is selected (rejected) in step i, it cannot be rejected (selected) in subsequent stages.

5.1 Optimal Storage on Tapes

Optimal Storage on Tapes is one of the applications in the Greedy Method. The objective of this algorithm is to find the Optimal retrieval time for accessing programs that are stored on tape.

This algorithm works in steps. In each step, it selects the best available options until all options are finished.

EXPLANATION

- There are 'n' programs that are to be stored on a computer tape of length (L).
- Associated with each program (i) is a length (l).
- Let the programs are stored in the order (I = i1, i2, i3, ….)

$$\sum_{k=1}^{j} l_{ik}$$

If all the programs retrieved often the **Expected or Mean Retrieval Time (MRT)** is

$$MRT = \frac{1}{n}\sum_{j=1}^{n} t_j$$

- If all programs are retrieved equally often then the expected or Mean Retrieval Time (MRT) is,

$$d(I) = \sum_{j=1}^{n}\sum_{k=1}^{j} l_{ik}$$

```
// Java Program to find the order of programs for which MRT is minimized
import java.io.*;
import java .util.*;
```

```
class GFG
{
// This functions outputs the required order and Minimum Retrieval Time
static void findOrderMRT(int []L, int n)
{     // Here length of i'th program is L[i]
      Arrays.sort(L);
      System.out.print("Optimal order in which " +
                  "programs are to be stored is: ");
      // Write code Here
           ………………….
      // MRT - Minimum Retrieval Time
      double MRT = 0;
      for (int i = 0; i < n; i++)
      {
            // Write code Here
           ………………….
      }
      System.out.print( "Minimum Retrieval Time" +
                              " of this order is " + MRT);
}
// Driver Code
public static void main (String[] args)
{
      int []L = { 2, 5, 4 };
      int n = L.length;
      findOrderMRT(L, n);
}
}
```

5.2 Job sequencing with deadlines

Job scheduling algorithm is applied to schedule the jobs on a single processor to maximize the profits.

The greedy approach of the job scheduling algorithm states that, "Given 'n' number of jobs with a starting time and ending time, they need to be scheduled in such a way that maximum profit is received within the maximum deadline".

Job Scheduling Algorithm:

Set of jobs with deadlines and profits are taken as an input with the job scheduling algorithm and scheduled subset of jobs with maximum profit are obtained as the final output.

Step1: Find the maximum deadline value from the input set of jobs.
Step2: Once, the deadline is decided, arrange the jobs in descending order of their profits.
Step3: Selects the jobs with highest profits, their time periods not exceeding the maximum deadline.
Step4: The selected set of jobs are the output.

```
// Java code for the above problem
import java.util.*;
class Job {
      // Each job has a unique-id,profit and deadline
      char id;
      int deadline, profit;
      // Constructors
      public Job() {}
      public Job(char id, int deadline, int profit)
      {     this.id = id;
            this.deadline = deadline;
            this.profit = profit;
      }
      // Function to schedule the jobs take 2 arguments
      // arraylist and no of jobs to schedule
      void printJobScheduling(ArrayList<Job> arr, int t)
      {
            //  Write Code Here
      ……. …….     ……
            }
      }

      // Driver's code
      public static void main(String args[])
      {
            ArrayList<Job> arr = new ArrayList<Job>();
            arr.add(new Job('a', 2, 100));
            arr.add(new Job('b', 1, 19));
            arr.add(new Job('c', 2, 27));
            arr.add(new Job('d', 1, 25));
            arr.add(new Job('e', 3, 15));

      System.out.println("Following is maximum profit sequence of jobs");
            Job job = new Job();
            // Function call
            job.printJobScheduling(arr, 3);
      }
}
```

5.3 Single source shortest path - Dijkstra's Algorithm

Dijkstra's Algorithm is also known as Single Source Shortest Path (SSSP) problem. It is used to find the shortest path from source node to destination node in graph.

The graph is widely accepted data structure to represent distance map. The distance between cities effectively represented using graph.

- Dijkstra proposed an efficient way to find the single source shortest path from the weighted graph. For a given source vertex s, the algorithm finds the shortest path to every other vertex v in the graph.
- Assumption: Weight of all edges is non-negative.
- Steps of the Dijkstra's algorithm are explained here:

1. Initializes the distance of source vertex to zero and remaining all other vertices to infinity.
2. Set source node to current node and put remaining all nodes in the list of unvisited vertex list. Compute the tentative distance of all immediate neighbour vertex of the current node.

3. If the newly computed value is smaller than the old value, then update it.
4. Weight updating in Dijkstra's Algorithm: When all the neighbours of a current node are explored, mark it as visited. Remove it from unvisited vertex list. Mark the vertex from unvisited vertex list with minimum distance and repeat the procedure.
5. Stop when the destination node is tested or when unvisited vertex list becomes empty.

```
// A Java program for Dijkstra's single source shortest path algorithm.
// The program is for adjacency matrix representation of the graph
import java.io.*;
import java.lang.*;
import java.util.*;
class ShortestPath {
      static final int V = 9;
      int minDistance(int dist[], Boolean sptSet[])
      {     // Initialize min value
            // Write Code Here
      …………….     ………..
      }
      // A utility function to print the constructed distance array
      void printSolution(int dist[])
      {     // Write Code Here
      }
      // Function that implements Dijkstra's single source shortest path
// algorithm for a graph represented using adjacency matrix representation
      void dijkstra(int graph[][], int src)
      {     int dist[] = new int[V];
// The output array. dist[i] will hold the shortest distance from src to i
      }
// Distance of source vertex from itself is always 0
      // Write Code Here
      ………..     …………
// Driver's code
      public static void main(String[] args)
      {  // Write Code Here
      ………..     …………
          // Function call
            t.dijkstra(graph, 0);
      }
}
```

5.4 Minimum spanning trees

A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree.
For a given Infix expression, convert it into Postfix form.
The minimum spanning tree from a graph is found using the following algorithms:

1. Prim's Algorithm
2. Kruskal's Algorithm

Prim's Algorithm
Prim's Algorithm begins with a single Node and adds up adjacent nodes one by one by discovering all of the connected edges along the way. Edges with the lowest weights that don't generate cycles are

chosen for inclusion in the MST structure. As a result, we can claim that Prim's algorithm finds the globally best answer by making locally optimal decisions.

Steps involved in Prim’s algorithms are mentioned below:

Step 1: Choose any vertex as a starting vertex.
Step 2: Pick an edge connecting any tree vertex and fringe vertex (adjacent vertex to visited vertex) having the minimum edge weight.
Step 3: Add the selected edge to MST only if it doesn't form any closed cycle.
Step 4: Keep repeating steps 2 and 3 until the fringe vertices exist.
Step 5: End.

```
// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.
import java.io.*;
import java.lang.*;
import java.util.*;
class MST {
      // Number of vertices in the graph
      private static final int V = 5;
      // Write Code Here
      …..   …..   ….
      void printMST(int parent[], int graph[][])
      {
            // Write Code Here
      …..   …..   ….
      }

      void primMST(int graph[][])
      {
            // Write Code Here
      …..   …..   ….          }
}
public static void main(String[] args)
      {     MST t = new MST();
            int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
            // Write Code Here
      …..   …..   ….
            t.primMST(graph);
      }
}
```

Kruskal’s Algorithm

Kruskal's approach sorts all the edges in ascending order of edge weights and only adds nodes to the tree if the chosen edge does not form a cycle. It also selects the edge with the lowest cost first and the edge with the highest cost last. As a result, we can say that the Kruskal’s Algorithm makes a locally optimum decision in the hopes of finding the global optimal solution. Hence, this algorithm can also be considered as a Greedy Approach.

The steps involved in Kruskal’s algorithm to generate a minimum spanning tree are:

Step 1: Sort all edges in increasing order of their edge weights.
Step 2: Pick the smallest edge.
Step 3: Check if the new edge creates a cycle or loop in a spanning tree.
Step 4: If it doesn’t form the cycle, then include that edge in MST. Otherwise, discard it.
Step 5: Repeat from step 2 until it includes |V| - 1 edges in MST.

```
// Java program for Kruskal's algorithm
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
public class KruskalsMST {
      // Defines edge structure
      static class Edge {
            // Write Code Here
      ….    …     ….
      }
      // Defines subset element structure
      static class Subset {
            // Write Code Here
      ….    …     ….
      }
      // Starting point of program execution
      public static void main(String[] args)
      { // Write Code Here
      ….    …     …. };
            kruskals(V, graphEdges);
      }

      // Function to find the MST
      private static void kruskals(int V, List<Edge> edges)
      { // Write Code Here
      ….    …     ….
            }
system.out.println( "Following are the edges of the constructed MST:");
            int minCost = 0;
            // Write Code Here
      ….    …     ….

      }
      // Function to unite two disjoint sets
      private static void union(Subset[] subsets, int x, int y)
      { // Write Code Here
      ….    …     ….
      }
      // Function to find parent of a set
      private static int findRoot(Subset[] subsets, int i)
      {
            if (subsets[i].parent == i)
                  return subsets[i].parent;

            subsets[i].parent
                  = findRoot(subsets, subsets[i].parent);
            return subsets[i].parent;
      }
}
```

5.5 Huffman coding

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
The variable-length codes assigned to input characters are Prefix coding, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.
Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is the prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be "cccd" or "ccb" or "acd" or "ab".

There are mainly two major parts in Huffman Coding

1. Build a Huffman Tree from input characters.
2. Traverse the Huffman Tree and assign codes to characters.

Algorithm:
The method which is used to construct optimal prefix code is called **Huffman coding**.

This algorithm builds a tree in bottom up manner. We can denote this tree by **T**
Let, |c| be number of leaves |c| -1 are number of operations required to merge the nodes. Q be the priority queue which can be used while constructing binary heap.

```
Algorithm Huffman (c)
{
  n= |c|
  Q = c
  for i<-1 to n-1
  do
  {
     temp <- get node ()

    left (temp] Get_min (Q) right [temp] Get Min (Q)
    a = left [templ b = right [temp]
    F [temp]<- f[a] + [b]
    insert (Q, temp)
   }
return Get_min (0)
}
```

Steps to build Huffman Tree
Input is an array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete.
   Let us understand the algorithm with an example:

```
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Scanner;
class Huffman {
// recursive function to print the huffman-code through the tree raversal.
// Here s is the huffman - code generated.
      public static void printCode(HuffmanNode root, String s)
      { // Write Code Here
            …     …     …
      }
// main function
      public static void main(String[] args)
      {     Scanner s = new Scanner(System.in);
            int n = 6;
            char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
            int[] charfreq = { 5, 9, 12, 13, 16, 45 };
            PriorityQueue<HuffmanNode>
            Q = new PriorityQueue<HuffmanNode>(n, new MyComparator());
            for (int i = 0; i < n; i++) {
                  // Write Code Here
                        …     ….    …
            }
            // create a root node
            HuffmanNode root = null;

            // Write Code Here
                        …     ….    …}
}
//node class is the basic structure of each node present in the Huffman -
class HuffmanNode {
      int data;
      char c;
      HuffmanNode left;
      HuffmanNode right;
}
class MyComparator implements Comparator<HuffmanNode> {
      public int compare(HuffmanNode x, HuffmanNode y)
      { // Write Code Here
                        …     ….    …     }
}
```

6. Greedy Technique

6.1 Assign Mice to Holes

Problem Statement and Example

Given the positions of mice and holes along a line, and the speed of each mouse, the task is to find the arrangement of mice in the holes that minimizes the maximum time taken for any mouse to reach its hole. Each mouse should be assigned to a unique hole, and the order of mice and holes should be preserved.

For example, consider the following scenario:

- Mice Positions: -3, 7, 5, 9, 16
- Hole Positions: 1, 9, 15, 4, 14

The goal is to arrange the mice in the holes in a way that minimizes the maximum time taken for any mouse to reach its hole.

Idea to Solve the Problem

To solve this problem, we can follow these steps:

1. Sort the positions of mice and holes in ascending order.
2. Iterate through the sorted arrays and calculate the absolute difference between each mouse's position and its corresponding hole's position.
3. Track the maximum absolute difference during the iteration.
4. The maximum absolute difference represents the time it takes for the mouse to reach its hole in the optimal arrangement.

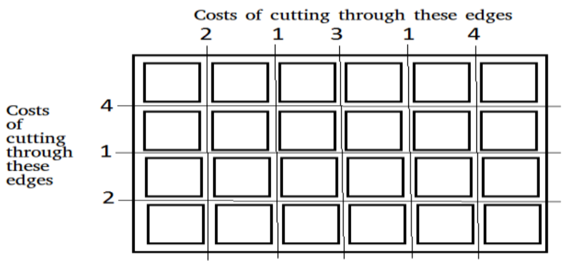
Algorithm Explanation

1. Sort both the mices and holes arrays in ascending order.
2. Initialize a variable result to store the maximum absolute difference (time taken).
3. Iterate through the arrays from 0 to n-1 (where n is the length of the arrays): a. Calculate the absolute difference between mices[i] and holes[i]. b. Update result if the calculated difference is greater than the current result.
4. Print the result, which represents the minimum time taken for the mice to reach their holes in an optimal arrangement.

```
// Java program to find the minimum time to place all mice in all holes.
import java.util.*;
public class GFG
{
      // Returns minimum time required to place mice in holes.
public int assignHole(ArrayList<Integer> mice,ArrayList <Integer> holes)
      {
            if (mice.size() != holes.size())
            return -1;
            /* Sort the lists */
            // Write Code Here
                  …     ….    …
            /* finding max difference between ith mice and hole */
      }
      /* Driver Function to test other functions */
      public static void main(String[] args)
      {
            GFG gfg = new GFG();
            ArrayList<Integer> mice = new ArrayList<Integer>();
            mice.add(4);
            mice.add(-4);
            mice.add(2);
            ArrayList<Integer> holes= new ArrayList<Integer>();
            holes.add(4);
            holes.add(0);
            holes.add(5);
            System.out.println("The last mouse gets into "+
            "the hole in time: "+gfg.assignHole(mice, holes));
      }
}
```

6.2 Minimum Cost to cut a board into squares

A board of length m and width n is given, we need to break this board into m*n squares such that cost of breaking is minimum. cutting cost for each edge will be given for the board. In short, we need to choose such a sequence of cutting such that cost is minimized.
Examples:



Costs of cutting through these edges
2
1
3
1
4
Costs of cutting through these edges
4
1
2

For above board optimal way to cut into square is:

Total minimum cost in above case is 42. It is

evaluated using following steps.

Initial Value : Total_cost = 0

Total_cost = Total_cost + edge_cost * total_pieces

Cost 4 Horizontal cut	Cost = 0 + 4*1 = 4
Cost 4 Vertical cut	Cost = 4 + 4*2 = 12
Cost 3 Vertical cut	Cost = 12 + 3*2 = 18
Cost 2 Horizontal cut	Cost = 18 + 2*3 = 24
Cost 2 Vertical cut	Cost = 24 + 2*3 = 30
Cost 1 Horizontal cut	Cost = 30 + 1*4 = 34
Cost 1 Vertical cut	Cost = 34 + 1*4 = 38
Cost 1 Vertical cut	Cost = 38 + 1*4 = 42

Minimum Cost to cut a board into squares using a greedy algorithm:

This problem can be solved by greedy approach . To get minimum cost , the idea is to cut the edge with highest cost first because we have less number of pieces and after every cut the number of pieces increase . As the question stated T**otal_cost = Total_cost + edge_cost * total_pieces .**

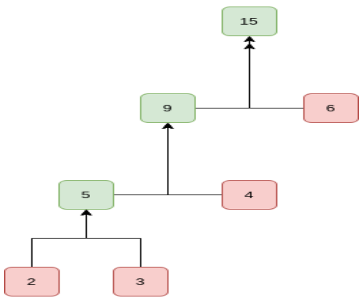
- At first sort both the array in non-ascending order
- We keep count of two variables vert(keeps track of vertical pieces) and **hzntl**(keeps track of horizontal pieces). We will initialize both of them with 1.
- We will keep track of two pointers starting from **0th** index of both the array
- Now we will take the highest cost edge from those pointers and multiply them with the corresponding variable. That is if we cut a horizontal cut we will **add (edge_cost*hzntl)** and increase vert by **1** and if we cut a vertical cut we will **add(edge_cost*vert)** and increase **hzntl** bt **1** .
- After cutting all the edges we will get the minimum cost

```
// Java program to divide a board into m*n squares
import java.util.Arrays;
import java.util.Collections;
class GFG
{
      // method returns minimum cost to break board into m*n squares
static int minimumCostOfBreaking(Integer X[], Integer Y[], int m, int n)
      {     int res = 0;
            // sort the horizontal cost in reverse order
            //Write Code here
            …..   …..   …..
            while (i < m && j < n)
            {//Write Code here
            …..   …..   …..

            }

            // loop for horizontal array, if remains
            //Write Code here
            …..   …..   …..

            // loop for vertical array,if remains
            //Write Code here
            …..   …..   …..
      }

      // Driver program
      public static void main(String arg[])
      {
            int m = 6, n = 4;
            Integer X[] = {2, 1, 3, 1, 4};
            Integer Y[] = {4, 1, 2};
            System.out.print(minimumCostOfBreaking(X, Y, m-1, n-1));
      }
}
```

6.3 Connect n ropes with minimum cost

Given are N ropes of different lengths, the task is to connect these ropes into one rope with minimum cost, such that the cost to connect two ropes is equal to the sum of their lengths.
Examples:
Input: arr[] = {4,3,2,6} , N = 4
Output: 29
Explanation:

1. First, connect ropes of lengths 2 and 3. Now we have three ropes of lengths 4, 6, and 5.
2. Now connect ropes of lengths 4 and 5. Now we have two ropes of lengths 6 and 9.
3. Finally connect the two ropes and all ropes have connected.



Input: arr[] = {1, 2, 3} , N = 3
Output: 9
Explanation:

1. First, connect ropes of lengths 1 and 2. Now we have two ropes of lengths 3 and 3.
2. Finally connect the two ropes and all ropes have connected.

Approach: If we observe the above problem closely, we can notice that the lengths of the ropes which are picked first are included more than once in the total cost. Therefore, the idea is to connect the smallest two ropes first and recur for the remaining ropes. This approach is similar to Huffman Coding. We put the smallest ropes down the tree so they can be repeated multiple times rather than the longer ones.

Algorithm: Follow the steps mentioned below to implement the idea:

- Create a min-heap and insert all lengths into the min-heap.
- Do following while the number of elements in min-heap is greater than one.
  - Extract the minimum and second minimum from min-heap
  - Add the above two extracted values and insert the added value to the min-heap.
  - Maintain a variable for total cost and keep incrementing it by the sum of extracted values.
- Return the value of total cost.

```
// Java program to connect n ropes with minimum cost
import java.util.*;
class ConnectRopes {
      static int minCost(int arr[], int n)
      {// Create a priority queue
            PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
      // Write Code Here
      …     ….    ……
            return res;
      }

      // Driver program to test above function
      public static void main(String args[])
      {
            int len[] = { 4, 3, 2, 6 };
            int size = len.length;
            System.out.println("Total cost for connecting"
                                    + " ropes is "
                                    + minCost(len, size));
      }
}
```

6.4 Rearrange a string so that all same characters become d distance away

Given a string and a positive integer d. Some characters may be repeated in the given string. Rearrange characters of the given string such that the same characters become d distance away from each other. Note that there can be many possible rearrangements, the output should be one of the possible rearrangements. If no such arrangement is possible, that should also be reported.
The expected time complexity is O(n + m log(MAX)) Here n is the length of string, m is the count of distinct characters in a string and MAX is the maximum possible different characters.

Examples:
Input: "abb", d = 2
Output: "bab"
Input: "aacbbc", d = 3
Output: "abcabc"
Input: "aaa", d = 2
Output: Cannot be rearranged

The approach to solving this problem is to count frequencies of all characters and consider the most frequent character first and place all occurrences of it as close as possible. After the most frequent character is placed, repeat the same process for the remaining characters.

1. Let the given string be str and size of string be n

2. Traverse str, store all characters and their frequencies in a Max Heap MH(implemented using priority queue). The value of frequency decides the order in MH, i.e., the most frequent character is at the root of MH.
3. Make all characters of str as '\0'.
4. Do the following while MH is not empty.
   - Extract the Most frequent character. Let the extracted character be x and its frequency be f.
   - Find the first available position in str, i.e., find the first '\0' in str.
   - Let the first position be p. Fill x at p, p+d,.. p+(f-1)d

```
// Java program to rearrange a string so that all same
// characters become atleast d distance away
import java.util.*;
class GFG
{
static int MAX_CHAR = 256;

// The function returns next eligible character with maximum frequency
// (Greedy!!) and zero or negative distance
static int nextChar(int freq[], int dist[])
{
      // Write code Here
      ….    …..   …
}
// The main function that rearranges input string 'str'
static int rearrange(char str[], char out[], int d)
{
      int n = str.length;

// Create an array to store all characters and their frequencies in str[]
      int []freq = new int[MAX_CHAR];

      // Write code Here
      ….    …..   …

      return 1;
}

// Driver code
public static void main(String[] args)
{
      char str[] = "aaaabbbcc".toCharArray();
      int n = str.length;

      // To store output
      char []out = new char[n];

      if (rearrange(str, out, 2)==1)
            System.out.println(String.valueOf(out));
      else
            System.out.println("Cannot be rearranged");
}
}
```

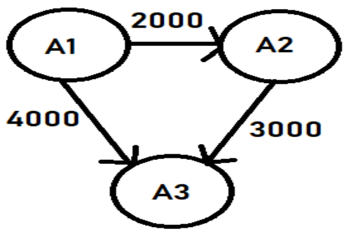
6.5 Minimization of Cash Flow among a given set of friends

Problem Statement

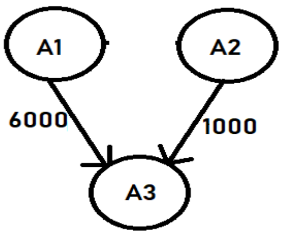
Suppose you are given that there are few friends. They have borrowed money from each other due to which there will be some cash flow on the network. Our main aim is to design an algorithm by which the total cash flow among all the friends is minimized.

For example:

There are three friends A1, A2, A3; you have to show the settlement of input debts between them.



A1 has to pay 2000 to A2 and 4000 to A3, A2 has to pay 3000 to A3



we can minimize the flow between A1 and A2 by directly paying to A3.

Solution using Greedy Approach

The greedy approach is used to build the solution in pieces, and this is what we want to minimize the cash flow. At every step, we will settle all the amounts of one person and recur for the remaining n-1 persons.

Calculate the net amount for every person, which can be calculated by subtracting all the debts, i.e., the amount to be paid from all credit, i.e., the amount to be paid to him. After this, we will find two persons with the maximum and the minimum net amounts. The person with a minimum of two is our first person to be settled and removed from the list.

Following algorithm will be done for every person varying 'i' from 0 to n-1.

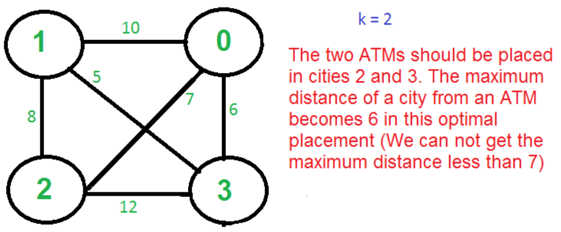
1. The first step will be to calculate the net amount for every person and store it in an amount array.
   Net amount = sum(received money) - sum(sent money).
2. Find the two people that have the most credit and the most debt. Let the maximum amount to be credited from maximum creditor be max_credit and the maximum amount to be debited from maximum debtor be max_debit. Let the maximum debtor be debt and maximum creditor be cred.
3. Let the minimum of two amounts be 'y'.
4. If y equals max_credit, delete cred from the list and repeat for the remaining (n-1) people.
5. If y equals max_debit, delete debt from the group of people and repeat for recursion.
6. If the amount is 0, then the settlement is done.

```
// Java program to find maximum cash  flow among a set of persons

class GFG
{
        // Number of persons (or vertices in the graph)
        static final int N = 3;

        // A utility function that returns  index of minimum value in arr[]
        static int getMin(int arr[])
        {
                int minInd = 0;
                for (int i = 1; i < N; i++)
                        if (arr[i] < arr[minInd])
                                minInd = i;
                return minInd;
        }

        // A utility function that returns index of maximum value in arr[]
        static int getMax(int arr[])
        {
                // Write Code Here
                …..     ……      ….
        }
        // A utility function to return minimum of 2 values
        static int minOf2(int x, int y)
        {
                return (x < y) ? x: y;
        }

        static void minCashFlowRec(int amount[])
        {
                // Write Code Here
                …..     ……      ….
        }

        static void minCashFlow(int graph[][])
        {// Write Code Here
                …..     ……      ….
        }
        // Driver code
        public static void main (String[] args)
        {
                // graph[i][j] indicates the amount  that person i needs to pay person j
                int graph[][] = { {0, 1000, 2000}, {0, 0, 5000}, {0, 0, 0},};
                minCashFlow(graph);
        }
}
```

6.6 Greedy Approximate Algorithm for K Centres Problem

Given n cities and distances between every pair of cities, select k cities to place warehouses (or ATMs or Cloud Server) such that the maximum distance of a city to a warehouse (or ATM or Cloud Server) is minimized.

For example consider the following four cities, 0, 1, 2, and 3, and the distances between them, how to place 2 ATMs among these 4 cities so that the maximum distance of a city to an ATM is minimized.


k = 2
The two ATMs should be placed in cities 2 and 3. The maximum distance of a city from an ATM becomes 6 in this optimal placement (We can not get the maximum distance less than 7)
10
5
8
7
6
12
1
0
2
3

There is no polynomial-time solution available for this problem as the problem is a known NP-Hard problem. There is a polynomial-time Greedy approximate algorithm, the greedy algorithm provides a solution that is never worse than twice the optimal solution. The greedy solution works only if the distances between cities follow Triangular Inequality (The distance between two points is always smaller than the sum of distances through a third point).

Approximate Greedy Algorithm:

1. Choose the first centre arbitrarily.
2. Choose remaining k-1 centres using the following criteria.
    - Let c1, c2, c3, … ci be the already chosen centres. Choose
    - (i+1)'th centre by picking the city which is farthest from already
    - selected centres, i.e, the point p which has following value as maximum
    - Min[dist(p, c1), dist(p, c2), dist(p, c3), …. dist(p, ci)]

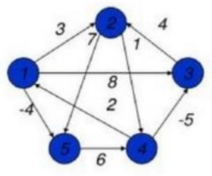
```
// Java program for the above approach
import java.util.*;
class GFG{
static int maxindex(int[] dist, int n)
{
      // Write Code Here
      ….. …..  ….
}
static void selectKcities(int n, int weights[][], int k)
{
      // Write Code Here
      ….. …..  ….

      }


      System.out.println(dist[max]);

      // Write Code Here
      ….. …..  ….

}

// Driver Code
public static void main(String[] args)
{
      int n = 4;
      int[][] weights = new int[][]{ { 0, 4, 8, 5 },  { 4, 0, 10, 7 },
                                    { 8, 10, 0, 9 },  { 5, 7, 9, 0 } };
      int k = 2;

      // Function Call
      selectKcities(n, weights, k);
}
}
```

7. Dynamic Programming

Dynamic Programming (DP) is defined as a technique that solves some particular type of problems in Polynomial Time. Dynamic Programming solutions are faster than the exponential brute method and can be easily proved their correctness. Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.
Irrevocable: Once the choice is made, it cannot be altered, i.e. if a feasible solution is selected (rejected) in step i, it cannot be rejected (selected) in subsequent stages.

7.1 All pairs Shortest Paths

The all pairs shortest path algorithm is also known as Floyd-Warshall algorithm is used to find all pair shortest path problem from a given Weighted Graph. As a result of this Algorithm , it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.



$$\begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

At first the output matrix is same as given cost matrix of the graph. After that the output matrix will be updated with all vertices k as the intermediate vertex.

The time complexity of this algorithm is O(V3), here V is the number of **vertices in the graph**.

Input − The cost matrix of the graph.

```
0 3 6 ∞ ∞ ∞ ∞
3 0 2 1 ∞ ∞ ∞
6 2 0 1 4 2 ∞
∞ 1 1 0 2 ∞ 4
∞ ∞ 4 2 0 2 1
∞ ∞ 2 ∞ 2 0 1
∞ ∞ ∞ 4 1 1 0
```

Output − Matrix of all pair shortest path.

```
0 3 4 5 6 7 7
3 0 2 1 3 4 4
```

```
4 2 0 1 3 2 3
5 1 1 0 2 3 3
6 3 3 2 0 2 1
7 4 2 3 2 0 1
7 4 3 3 1 1 0
```

Algorithm

floydWarshal(cost)

Input − The cost matrix of given Graph.

Output − Matrix to for shortest path between any vertex to any vertex.

```
Begin
  for k := 0 to n, do
    for i := 0 to n, do
      for j := 0 to n, do
        if cost[i,k] + cost[k,j] < cost[i,j], then
          cost[i,j] := cost[i,k] + cost[k,j]
        done
      done
    done
    display the current cost matrix
End
```

```java
// Java program for Floyd Warshall All Pairs Shortest Path algorithm.
import java.io.*;
import java.lang.*;
import java.util.*;
class AllPairShortestPath {
	final static int INF = 99999, V = 4;
	void floydWarshall(int dist[][])
	{
		// Write Code Here
		…..	……	…..
		printSolution(dist);
	}

	void printSolution(int dist[][])
	{
		System.out.println(
			"The following matrix shows the shortest "
			+ "distances between every pair of vertices");
		for (int i = 0; i < V; ++i) {
			for (int j = 0; j < V; ++j) {
				if (dist[i][j] == INF)
					System.out.print("INF ");
				else
					System.out.print(dist[i][j] + " ");
```

```
                  }
                  System.out.println();
            }
      }

      // Driver's code
      public static void main(String[] args)
      {
            // Provide Input
                  ….          …..   ….
            AllPairShortestPath a = new AllPairShortestPath();
            a.floydWarshall(graph);
      }
}
```

7.2 Optimal Binary Search Tree

An Optimal Binary Search Tree (OBST), also known as a Weighted Binary Search Tree, is a binary search tree that minimizes the expected search cost. In a binary search tree, the search cost is the number of comparisons required to search for a given key.

In an OBST, each node is assigned a weight that represents the probability of the key being searched for. The sum of all the weights in the tree is 1.0. The expected search cost of a node is the sum of the product of its depth and weight, and the expected search cost of its children.

To construct an OBST, we start with a sorted list of keys and their probabilities. We then build a table that contains the expected search cost for all possible sub-trees of the original list. We can use dynamic programming to fill in this table efficiently. Finally, we use this table to construct the OBST.

The time complexity of constructing an OBST is O(n^3), where n is the number of keys. However, with some optimizations, we can reduce the time complexity to O(n^2). Once the OBST is constructed, the time complexity of searching for a key is O(log n), the same as for a regular binary search tree.

The OBST is a useful data structure in applications where the keys have different probabilities of being searched for. It can be used to improve the efficiency of searching and retrieval operations in databases, compilers, and other computer programs.

Given a sorted array key *[0.. n-1]* of search keys and an array *freq[0.. n-1]* of frequency counts, where *freq[i]* is the number of searches for *keys[i]*. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.
Let us first define the cost of a BST. The cost of a BST node is the level of that node multiplied by its frequency. The level of the root is 1.

Examples:

Input: keys[] = {10, 12}, freq[] = {34, 50}

There can be following two possible BSTs

```
   10           12
    \          /
     12       10
    I         II
```

Frequency of searches of 10 and 12 are 34 and 50 respectively.

The cost of tree I is 34*1 + 50*2 = 134

The cost of tree II is 50*1 + 34*2 = 118

Input: keys[] = {10, 12, 20}, freq[] = {34, 8, 50}

There can be following possible BSTs

```
  10          12           20      10         20
   \         /  \         /          \        /
    12      10   20      12           20     10
     \                  /            /         \
      20               10           12          12
  I          II        III         IV         V
```

Among all possible BSTs, cost of the fifth BST is minimum.

Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142

We can use the recursive solution with a dynamic programming approach to have a more optimized code, reducing the complexity from O(n^3) from the pure dynamic programming to O(n). To do that, we have to store the subproblems calculations in a matrix of NxN and use that in the recursions, avoiding calculating all over again for every recursive call.

```
// Dynamic Programming Java code for Optimal Binary Search Tree Problem
public class Optimal_BST {

      /* A Dynamic Programming based function that calculates
            minimum cost of a Binary Search Tree. */
      static int optimalSearchTree(int keys[], int freq[], int n) {

      /* Create an auxiliary 2D matrix to store results of subproblems */
            int cost[][] = new int[n + 1][n + 1];
             //  Write Code Here
            ….    ….    …}

// A utility function to get sum of array elements freq[i] to freq[j]
      static int sum(int freq[], int i, int j) {
            int s = 0;]
            //  Write Code Here
              ….. ….    …  }
      public static void main(String[] args) {

            int keys[] = { 10, 12, 20 };
            int freq[] = { 34, 8, 50 };
            int n = keys.length;
            System.out.println("Cost of Optimal BST is "
                        + optimalSearchTree(keys, freq, n));
      }

}
```

7.3 0/1 Knapsack Problem

Given **N** items where each item has some weight and profit associated with it and also given a bag with capacity **W**, [i.e., the bag can hold at most **W** weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible. Stop when the destination node is tested or when unvisited vertex list becomes empty.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

<u>Memoization Approach for 0/1 Knapsack Problem:</u>

Note: It should be noted that the above function using recursion computes the same subproblems again and again. See the following recursion tree, K(1, 1) is being evaluated twice.

In the following recursion tree, **K()** refers to knapSack(). The two parameters indicated in the following recursion tree are n and W.
The recursion tree is for following sample inputs.
weight[] = {1, 1, 1}, W = 2, profit[] = {10, 20, 30}

```
                              K(3, 2)
                          /              \
                        /                  \
               K(2, 2)                      K(2, 1)
            /          \                  /          \
          /              \              /              \
     K(1, 2)           K(1, 1)       K(1, 1)       K(1, 0)
      /   \             /   \           /               \
    /       \         /       \       /                   \
K(0, 2)  K(0, 1)  K(0, 1)  K(0, 0)  K(0, 1)          K(0, 0)
```

Recursion tree for Knapsack capacity 2 units and 3 items of 1 unit weight.

As there are repetitions of the same subproblem again and again we can implement the following idea to solve the problem.

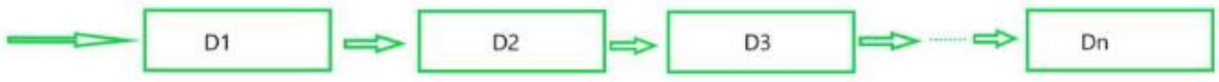
If we get a subproblem the first time, we can solve this problem by creating a 2-D array that can store a particular state (n, w). Now if we come across the same state (n, w) again instead of calculating it in exponential complexity we can directly return its result stored in the table in constant time.

```
// Dynamic Programming Java code for Optimal Binary Search Tree Problem
public class Optimal_BST {
      static int optimalSearchTree(int keys[], int freq[], int n) {
            int cost[][] = new int[n + 1][n + 1];
                  // Write Code Here
            ….    …..   ….

      }
      static int sum(int freq[], int i, int j) {
            int s = 0;
            for (int k = i; k <= j; k++) {
                  if (k >= freq.length)
                        continue;
                  s += freq[k];
            }
            return s;
      }

      public static void main(String[] args) {
            int keys[] = { 10, 12, 20 };
            int freq[] = { 34, 8, 50 };
            int n = keys.length;
            System.out.println("Cost of Optimal BST is "
                        + optimalSearchTree(keys, freq, n));
      }

}
```

7.4 Reliability Design

The reliability design problem is the designing of a system composed of several devices connected in series or parallel. **Reliability** means the probability to get the success of the device.

Let's say, we have to set up a system consisting of **D_1, D_2, D_3, …………, and D_n** devices, each device has some costs **C_1, C_2, C_3, ……., C_n.** Each device has a reliability of **0.9** then the entire system has reliability which is equal to the **product** of the reliabilities of all devices i.e., **$\pi r_i = (0.9)^4$.**



Serial Combination of one copy of each device

It means that **35%** of the system has a chance to fail, due to the failure of any one device. the problem is that we want to construct a system whose reliability is maximum. How it can be done so? we can think that we can take more than one copy of each device so that if one device fails we can use the copy of that device, which means we can connect the devices **parallel**.

When the same type of **3** devices is connected parallelly in **stage 1** having a reliability 0.9 each then:

```
public class SeriesSystemReliability {
    public static void main(String[] args) {
        double[] componentReliabilities = {0.99, 0.98, 0.97, 0.96, 0.95};
        double systemReliability = 
calculateSeriesSystemReliability(componentReliabilities);
        System.out.println("The reliability of the series system is: " + 
systemReliability);
    }
    public static double calculateSeriesSystemReliability(double[] 
reliabilities) {
        double systemReliability = 1.0; // Start with a system that's 
initially 100% reliable
        for (double reliability : reliabilities) {
            systemReliability *= reliability;
        }
        return systemReliability;
    }
}
```

7.5 Flow Shop Scheduling

Flow shop scheduling problem:

- In flow shop, m different machines should process n jobs. Each job contains exactly n operations. The i^th^ operation of the job must be executed on the i^th^ machine. Operations within one job must be performed in the specified order.
- The first operation gets executed on the first machine, then the second operation on the second machine, and so on. Jobs can be executed in any order. The problem is to determine the optimal such arrangement, i.e. the one with the shortest possible total job execution makespan.
- With two machines, problem can be solved in O(nlogn) time using Johnson's algorithm. For more than 2 machines, the problem is NP hard. The goal is to minimize the sum of completion time of all jobs.
- The flow shop contains n jobs simultaneously available at time zero and to be processed by two machines arranged in series with unlimited storage in between them. The processing time of all jobs are known with certainty.
- It is required to schedule n jobs on machines so as to minimize makespan. The Johnson's rule for scheduling jobs in two machine flow shop is given below: In an optimal schedule, job i precedes job j if $\min\{p_{i1},p_{j2}\} < \min\{p_{j1},p_{i2}\}$. Whereas, p_{i1} is the processing time of job i on machine 1 and p_{i2} is the processing time of job i on machine 2. Similarly, p_{j1} and p_{j2} are processing times of job j on machine 1 and machine 2 respectively.
- We can find the optimal scheduling using Dynamic Programming.

  Algorithm for Flow Shop Scheduling

  Johnson's algorithm for flow shop scheduling is described below:

```
Algorithm JOHNSON_FLOWSHOP(T, Q)
// T is array of time of jobs, each column indicating time on machine Mi
// Q is queue of jobs

Q = Φ

for j = 1 to n do
    t = minimum machine time scanning in booth columns
    if t occurs in column 1 then
        Add Job j to the first empty slot of Q
    else
        Add Job j  to last empty slot of Q
    end
    Remove processed job from consideration
end
return Q
```

```
public class FlowshopScheduling {
    static class Job {
        int jobID;
        int timeMachine1;
        int timeMachine2;
        public Job(int jobID, int timeMachine1, int timeMachine2) {
            this.jobID = jobID;
            this.timeMachine1 = timeMachine1;
            this.timeMachine2 = timeMachine2;
        }
    }

    public static void main(String[] args) {
        Job[] jobs = {
            new Job(1, 3, 2),
            new Job(2, 2, 1),
            new Job(3, 4, 3)
        };

        // Write Code Here
      ..    …     ..    ….    ….

        System.out.println("Minimum makespan: " + makespan);
    }
}
```

8. Dynamic Programming

8.1 Sherlock and Cost

Problem Statement and Example

In this challenge, you will be given an array B and must determine an array A. There is a special rule: For all i, A[i]<=B[i]. That is, A[i] can be any number you choose such that 1<=A[i]<=B[i]. Your task is to select a series of A[i] given B[i] such that the sum of the absolute difference of consecutive pairs of A is maximized. This will be the array's cost and will be represented by the variable S below.

The equation can be written:

$$S = \sum_{i=2}^{n} |A[i] - A[i-1]|$$

For example, if the array B=[1,2,3], we know that 1<=A[1]<=1, 1<=A[2]<=2, and 1<=A[3]<=3. Arrays meeting those guidelines are:
[1,1,1], [1,1,2], [1,1,3]
[1,2,1], [1,2,2], [1,2,3]

Our calculations for the arrays are as follows:
|1-1| + |1-1| = 0 |1-1| + |2-1| = 1 |1-1| + |3-1| = 2
|2-1| + |1-2| = 2 |2-1| + |2-2| = 1 |2-1| + |3-2| = 2
The maximum value obtained is 2.

Function Description:

Complete the cost function in the editor below. It should return the maximum value that can be obtained.
cost has the following parameter(s):
B: an array of integers

Input Format
The first line contains the integer t, the number of test cases.
Each of the next t pairs of lines is a test case where:
- The first line contains an integer n, the length of B
- The next line contains n space-separated integers B[i]
Constraints
1<=t<=20
1<n<=10^5
1<=B[i]<=100

Output Format
For each test case, print the maximum sum on a separate line.

Sample Input
1
5
10 1 10 1 10

Sample Output
36

Explanation
The maximum sum occurs when A[1]=A[3]=A[5]=10 and A[2]=A[4]=1. That is 36.

```
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;

int b[100000];
int dp[100000][2];

int main()
{
      int t;
      cin >> t;
      while (t--)
      {
      memset(dp, 0, sizeof(dp));
      int n;
      cin >> n;
      for (int i = 0; i < n; i++)
      cin >> b[i];
      for (int i = 1; i < n; i++)
      {
       dp[i][0] = max(dp[i - 1][0], dp[i - 1][1] + abs(b[i - 1] - 1));
       dp[i][1] = max(dp[i - 1][0] + abs(b[i] - 1), dp[i - 1][1]);
      }
      cout << max(dp[n - 1][0], dp[n - 1][1]) << endl;
      }
      return 0;
}
```

8.2 Summing Pieces

Consider an array, A, of length n. We can split A into contiguous segments called pieces and store them as another array, B. For example, if A=[1,2,3], we have the following arrays of pieces:

B=[(1),(2),(3)] contains three 1-element pieces.
B=[(1,2),(3)] contains two pieces, one having 2 elements and the other having 1 element.
B=[(1),(2,3)] contains two pieces, one having 1 element and the other having 2 elements.
B=[(1,2,3)] contains one 3-element piece.
We consider the value of a piece in some array B to be
(sum of all numbers in the piece) X (length of piece), and we consider the total value of some array B to be the sum of the values for all pieces in that B. For example, the total value of B=[(1,2,4), (5,1), (2)] is (1+2+4) X3+(5+1)X2+(2)X1=35.

Given A, find the total values for all possible B's, sum them together, and print this sum modulo (10^9 + 7) on a new line.
Input Format
The first line contains a single integer, n, denoting the size of array A.
The second line contains n space-separated integers describing the respective values in A.
Output Format
Print a single integer denoting the sum of the total values for all piece arrays (B's) of A, modulo (10^9 + 7).
Sample Input 1
3
1 3 6
Sample Output 1
73
Sample Input 2
5
4 2 9 10 1
Sample Output 2
971

```
public class SummingPieces {

    static final int MOD = 1000000007;

    public static void main(String[] args) {
        int[] arr = {1, 3, 6}; // Example array
        System.out.println("Total sum is: " + summingPieces(arr));
    }

    public static long summingPieces(int[] arr) {
        int n = arr.length;
        long totalSum = 0;
        long[] pow2 = new long[n];
        pow2[0] = 1; // 2^0
      // Write Code Here
      …     …..   ….
        }

        return totalSum;
    }
}
```

8.3 Stock Maximize

Given are N ropes of different lengths, the task is to connect these
Your algorithms have become so good at predicting the market that you now know what the share price of Wooden Orange Toothpicks Inc. (WOT) will be for the next number of days.

Each day, you can either buy one share of WOT, sell any number of shares of WOT that you own, or not make any transaction at all. What is the maximum profit you can obtain with an optimum trading strategy?

Example
Prices=[1,2]
Buy one share day one, and sell it day two for a profit of 1. Return 1.
Prices[2,1]
No profit can be made so you do not buy or sell stock those days. Return 0.

Function Description
Complete the stockmax function in the editor below.
stockmax has the following parameter(s):

- prices: an array of integers that represent predicted daily stock prices

Returns

- int: the maximum profit achievable

Input Format
The first line contains the number of test cases t.
Each of the next t pairs of lines contain:
- The first line contains an integer n, the number of predicted prices for WOT.
- The next line contains n space-separated integers prices[i], each a predicted stock price for day i.
Constraints

- 1<=t<=10

- 1<=n<=50000

- 1<=prices[i]<=100000

Output Format
Output i lines, each containing the maximum profit that can be obtained for the corresponding test case.
Sample Input
STDIN   Function
-----   --------

3	q = 3
3	prices[] size n = 3
5 3 2	prices = [5, 3, 2]
3	prices[] size n = 3
1 2 100	prices = [1, 2, 100]
4	prices[] size n = 4
1 3 1 2	prices =[1, 3, 1, 2]

Sample Output

```
0
197
3
```

Explanation

For the first case, there is no profit because the share price never rises, return 0.
For the second case, buy one share on the first two days and sell both of them on the third day for a profit of 197.
For the third case, buy one share on day 1, sell one on day 2, buy one share on day 3, and sell one share on day 4. The overall profit is 3.

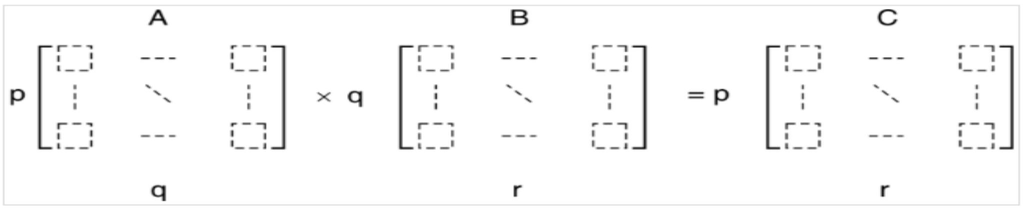
```java
public class StockMaximize {

    public static void main(String[] args) {
        int[] prices = {1, 2, 100};
        System.out.println("Maximum Profit: " + maxProfit(prices));
    }

    public static long maxProfit(int[] prices) {
        // Write Code Here
            ….    ….    …..
        }

        return maxProfit;
    }
```

8.4 Matrix Chain Multiplication Problem

Problem: In what order, n matrices A_1, A_2, A_3, …. A_n should be multiplied so that it would take a minimum number of computations to derive the result.

Cost of matrix multiplication: Two matrices are called compatible only if the number of columns in the first matrix and the number of rows in the second matrix are the same. Matrix multiplication is possible only if they are compatible. Let A and B be two compatible matrices of dimensions p x q and q x r

Each element of each row of the first matrix is multiplied with corresponding elements of the appropriate column in the second matrix. The total numbers of multiplications required to multiply matrix A and B are p x q x r.



Suppose dimension of three matrices are :
$A_1 = 5 \times 4$
$A_2 = 4 \times 6$
$A_3 = 6 \times 2$
Matrix multiplication is associative. So
$(A_1 A_2) A_3 = \{(5 \times 4) \times (4 \times 6)\} \times (6 \times 2)$
$= (5 \times 4 \times 6) + (5 \times 6 \times 2)$
$= 180$
$A_1 (A_2 A_3) = (5 \times 4) \times \{(4 \times 6) \times (6 \times 2)\}$
$= (5 \times 4 \times 2) + (4 \times 6 \times 2)$
$= 88$
The answer of both multiplication sequences would be the same, but the numbers of multiplications are different. This leads to the question, what order should be selected for a chain of matrices to minimize the number of multiplications?

Counting the number of paranthesization

Let us denote the number of alternative parenthesizations of a sequence of n matrices by p(n). When n = 1, there is only one matrix and therefore only one way to parenthesize the matrix. When n ≥ 2, a fully parenthesized matrix product is the product of two fully parenthesized matrix sub-products, and the split between the two subproducts may occur between the k and (k + 1)st matrices for any k = 1, 2, 3..., n – 1. Thus we obtain the recurrence.

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} p(k)\, p(n-k) & \text{if } n \geq 2 \end{cases}$$

The solution to the recurrence is the sequence of ***Catalan numbers***, which grows as $\Omega(4^n / n^{3/2})$, roughly equal to $\Omega(2^n)$. Thus, the numbers of solutions are exponential in n. A brute force attempt is infeasible to find the solution.

Any parenthesizations of the product $A_i A_{i+1} \ldots A_j$ must split the product between A_k and A_{k+1} for some integer k in the range $i \leq k < j$. That is for some value of k, we first compute the matrices $A_{i....k}$ and $A_{k+1...j}$ and then multiply them together to produce the final product $A_{i...j}$ The cost of computing these parenthesizations is the cost of computing $A_{i....k}$, plus the cost of computing $A_{k+1...j}$ plus the cost of multiplying them together.

We can define m[i, j] recursively as follows. If i == j, the problem is trivial; the chain consists of only one matrix $A_{i...i}$ = A. No scalar multiplications are required. Thus m[i, i] = 0 for i = 1, 2 ...n.

To compute m[i, j] when i < j, we take advantage of the structure of an optimal solution of the first step. Let us assume that the optimal parenthesizations split the product $A_i A_{i+1} \ldots A_j$ between A_k and A_{k+1}, where $i \le k < j$. Then m[i, j] is equal to the minimum cost for computing the subproducts $A_{i \ldots k}$ and $A_{k+1 \ldots j}$ plus the cost of multiplying these two matrices together.

The optimal substructure is defined as,

$$m[i,j] = \begin{cases} 0 & , \text{if } i = j \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + d_{i-1} \times d_k \times d_j\} & , \text{if } i < j \end{cases}$$

Where d = $\{d_0, d_1, d_2, \ldots, d_n\}$ is the vector of matrix dimensions.

m[i, j] = Least number of multiplications required to multiply matrix sequence $A_i \ldots A_j$.

```
// Java code to implement the matrix chain multiplication using recursion
import java.io.*;
import java.util.*;
class MatrixChainMultiplication {
      static int MatrixChainOrder(int p[], int i, int j)
      {
            // Write Code Here
      …….. ……….       ……….. 
      }
      // Driver code
      public static void main(String args[])
      {
            int arr[] = new int[] { 1, 2, 3, 4, 3 };
            int N = arr.length;

            // Function call
            System.out.println(
                  "Minimum number of multiplications is "
                  + MatrixChainOrder(arr, 1, N - 1));
      }
}
```

8.5 Bridge and Torch Problem

Problem Statement

Given an array of positive distinct integer denoting the crossing time of 'n' people. These 'n' people are standing at one side of bridge. Bridge can hold at max two people at a time. When two people cross the bridge, they must move at the slower person's pace. Find the minimum total time in which all persons can cross the bridge.

Note: Slower person space is given by larger time.

Input: Crossing Times = {10, 20, 30}

Output: 60

Explanation

1. Firstly person '1' and '2' cross the bridge
   with total time about 20 min(maximum of 10, 20)
2. Now the person '1' will come back with total
   time of '10' minutes.
3. Lastly the person '1' and '3' cross the bridge
   with total time about 30 minutes

Hence total time incurred in whole journey will be
20 + 10 + 30 = 60

Input: Crossing Times = [1, 2, 5, 8}
Output: 15
Explanation
The approach is to use Dynamic programming. Before getting dive into dynamic programming let's see the following observation that will be required in solving the problem.

1. When any two people cross the bridge, then the fastest person crossing time will not be contributed in answer as both of them move with slowest person speed.
2. When some of the people will cross the river and reached the right side then only the fastest people(smallest integer) will come back to the left side.
3. Person can only be present either left side or right side of the bridge. Thus, if we maintain the left mask, then right mask can easily be calculated by setting the bits '1' which is not present in the left mask. For instance, Right_mask = ((2^n) – 1) XOR (left_mask).
4. Any person can easily be represented by bitmask(usually called as 'mask'). When ith bit of 'mask' is set, that means that person is present at left side of the bridge otherwise it would be present at right side of bridge. For instance, let the mask of **6 people** is 100101, which represents the person 1, 4, 6 are present at left side of bridge and the person 2, 3 and 5 are present at the right side of the bridge.

```
//To find minimum time required to send people on other side of bridge
import java.io.*;
class GFG {
      static int dp[][] = new int[1 << 20][2];
      public static int findMinTime(int leftmask, boolean turn, int arr[],
                                                      int n)
      {     // If all people has been transferred
            if (leftmask == 0)
                  return 0;
            // Write Code Here
            ….    …..   …..

            else {      if (Integer.bitCount(leftmask) == 1) {
                        for (int i = 0; i < n; ++i) {
                       // Write Code Here
                        ….    …..   …..
                  }
            // Write Code Here
            ….    …..   …..}

      // Utility function to find minimum time
      public static int findTime(int arr[], int n)
```

```
        {
                // Write Code Here
                ….    …..   …..   }

        // Driver Code
        public static void main(String[] args)
        {
                int arr[] = { 10, 20, 30 };
                int n = 3;
                System.out.print(findTime(arr, n));
        }
}
```

9. BackTracking

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

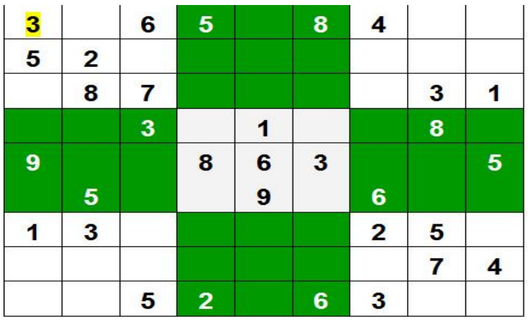
<u>Types of Backtracking Algorithm</u>

There are three types of problems in backtracking

1. Decision Problem: In this, we search for a feasible solution.
2. Optimization Problem: In this, we search for the best solution.
3. Enumeration Problem: In this, we find all feasible solutions.

9.1 Sudoku Puzzle

Given a partially filled 9×9 2D array 'grid[9][9]', the goal is to assign digits (from 1 to 9) to the empty cells so that every row, column, and subgrid of size 3×3 contains exactly one instance of the digits from 1 to 9.



3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

Sudoku using Backtracking:

Like all other BackTracking problem, Sudoku can be solved by assigning numbers one by one to empty cells. Before assigning a number, check whether it is safe to assign.

Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

Follow the steps below to solve the problem:

- Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.
- Create a recursive function that takes a grid.
- Check for any unassigned location.
    - If present then assigns a number from 1 to 9.
    - Check if assigning the number to current index makes the grid unsafe or not.
    - If safe then recursively call the function for all safe cases from 0 to 9.
    - If any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.
- If there is no unassigned location then return true.

```
/* A Backtracking program in Java to solve Sudoku problem */
class GFG
{
public static boolean isSafe(int[][] board, int row, int col, int num)
      {
            // Row has the unique (row-clash)
            for (int d = 0; d < board.length; d++)
            {     // Write Code Here
            ….    …..   …..}
            for (int r = 0; r < board.length; r++)
            {     // Write Code Here
            ….    …..   …..
            }
            // Write Code Here
            ….    …..   …..
      }

      public static boolean solveSudoku(int[][] board, int n)
      {
            // Write Code Here
            ….    ….    …
      }

      public static void print(int[][] board, int N)
      {
            for (int r = 0; r < N; r++)
            {  for (int d = 0; d < N; d++)
                  {
                        System.out.print(board[r][d]);
                        System.out.print(" ");
                  }
                  System.out.print("\n");

                  if ((r + 1) % (int)Math.sqrt(N) == 0)
                  {
                        System.out.print("");
                  }
            }
      }

      // Driver Code
      public static void main(String args[])
      {     // Write Code Here
            …..   …..   ….
            };
            int N = board.length;

            if (solveSudoku(board, N))
            {  print(board, N);           }
            else {
                  System.out.println("No solution");
            }
```

9.2 8-Queen Problem

The eight queens problem is the problem of placing eight queens on an 8×8 chessboard such that none of them attack one another (no two are in the same row, column, or diagonal). More generally, the n queens problem places n queens on an n×n chessboard. There are different solutions for the problem.

Explanation:

- This **pseudocode uses a backtracking algorithm** to find a solution to the 8 Queen problem, which consists of placing 8 queens on a chessboard in such a way that no two queens threaten each other.
- The algorithm starts by placing a queen on the first column, then it proceeds to the next column and places a queen in the **first safe** row of that column.
- If the algorithm reaches the 8th column and al**l queens are placed in a safe position**, it prints the board and returns true.
  If the algorithm is unable to place a queen in a safe position in a certain column, it backtracks to the previous column and tries a different row.
- The “isSafe” function **checks** if it is safe to place a queen on a certain row and column by checking if there are any queens in the same row, diagonal or anti-diagonal.
- It’s worth to notice that this is just a high-level **pseudocode** and it might need to be adapted depending on the specific implementation and language you are using.

```
N = 8 # (size of the chessboard)

def solveNQueens(board, col):
      if col == N:
            print(board)
            return True
      for i in range(N):
            // Write code here
            …     ….    …

def isSafe(board, row, col):
      // Write code here
            …     ….    …
      return True

board = [[0 for x in range(N)] for y in range(N)]
if not solveNQueens(board, 0):
      print("No solution found")
```

9.3 Sum of Subsets

Given a **set[]** of non-negative integers and a value **sum**, the task is to print the subset of the given set whose sum is equal to the given **sum**.

Examples:

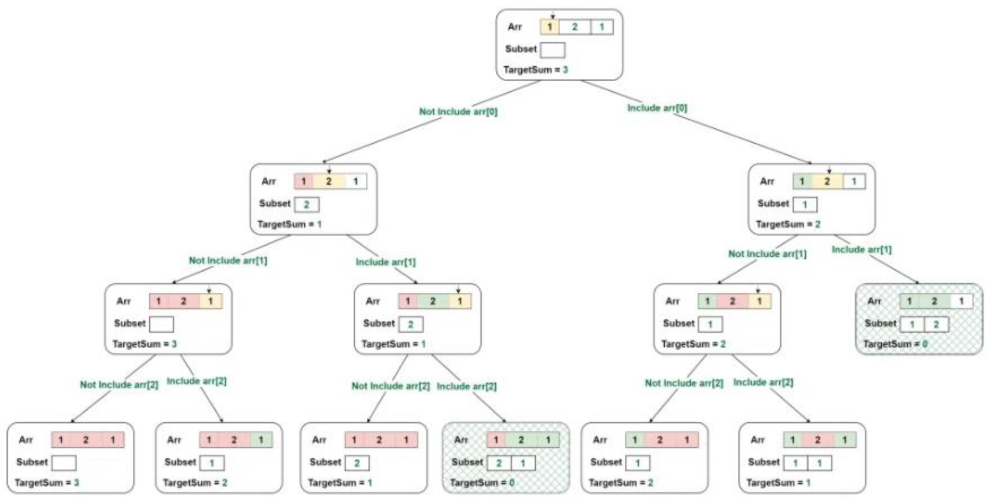
Input: set[] = {1,2,1}, sum = 3
Output: [1,2],[2,1]
Explanation: There are subsets [1,2],[2,1] with sum 3.

Input: set[] = {3, 34, 4, 12, 5, 2}, sum = 30
Output: []
Explanation: There is no subset that add up to 30.

ubset sum can also be thought of as a special case of the 0/1 knapsack problem. For each item, there are two possibilities:

- **Include** the current element in the subset and recur for the remaining elements with the remaining **Sum**.
- **Exclude** the current element from the subset and recur for the remaining elements.

Finally, if **Sum** becomes **0** then print the elements of current subset. The recursion's **base case** would be when **no items are left**, or the **sum** becomes **negative**, then simply return.



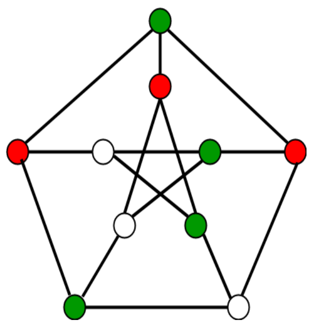
Subset Sum Problem

```
import java.util.ArrayList;
import java.util.List;
public class SubsetSum {
      static boolean flag = false;
      static void printSubsetSum(int i, int n, int[] set,int targetSum,
                                          List<Integer> subset)
      {// Write Code Here
      ….    ….    ….
            }

      // Driver code
      public static void main(String[] args)
      {
            // Test case 1
            int[] set1 = { 1, 2, 1 };
            int sum1 = 3;
            int n1 = set1.length;
            List<Integer> subset1 = new ArrayList<>();
            System.out.println("Output 1:");
            printSubsetSum(0, n1, set1, sum1, subset1);
            System.out.println();
            flag = false;

            // Test case 2
            int[] set2 = { 3, 34, 4, 12, 5, 2 };
            int sum2 = 30;
            int n2 = set2.length;
            List<Integer> subset2 = new ArrayList<>();
            System.out.println("Output 2:");
            printSubsetSum(0, n2, set2, sum2, subset2);
            if (!flag) {
                  System.out.println("There is no such subset");
            }
      }
}
```

9.4 Graph Coloring

Given an undirected graph and a number **m**, the task is to color the given graph with at most **m** colors such that no two adjacent vertices of the graph are colored with the same color

Note: Here coloring of a graph means the assignment of colors to all verticesBelow is an example of a graph that can be colored with 3 different colors:



Assign colors one by one to different vertices, starting from vertex 0. Before assigning a color, check for safety by considering already assigned colors to the adjacent vertices i.e check if the adjacent vertices have the same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the solution. If no assignment of color is possible then backtrack and return false

Follow the given steps to solve the problem:

- Create a recursive function that takes the graph, current index, number of vertices, and color array.
- If the current index is equal to the number of vertices. Print the color configuration in the color array.
- Assign a color to a vertex from the range (1 to m).
- For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) and recursively call the function with the next index and number of vertices otherwise, return **false**
- If any recursive function returns **true** then return **true**
- If no recursive function returns **true** then return **false**

```
/* Java program for solution of M Coloring problem using backtracking */

public class mColoringProblem {
        final int V = 4;
        int color[];

        /* A utility function to check if the current color assignment  is safe for vertex v */
        boolean isSafe(int v, int graph[][], int color[], int c)
        {
                //  Write Code Here
        ....        ....        ...
        }
        boolean graphColoringUtil(int graph[][], int m, int color[], int v)
        {
                /* base case: If all vertices are
                //  Write Code Here
        ....        ....        ...                                         }
                }
                return false;
        }

        boolean graphColoring(int graph[][], int m)
        {//  Write Code Here
        ....        ....        ...                     }

        void printSolution(int color[])
        {//  Write Code Here
        ....        ....        ...                     }

        // Driver code
        public static void main(String args[])
        { //   Write Code Here to input Graph
        ....        ....        ...
                Coloring.graphColoring(graph, m);
        }
}
```

9.5 Hamiltonian Cycle

Hamiltonian Cycle or Circuit in a graph **G** is a cycle that visits every vertex of **G** exactly once and returns to the starting vertex.

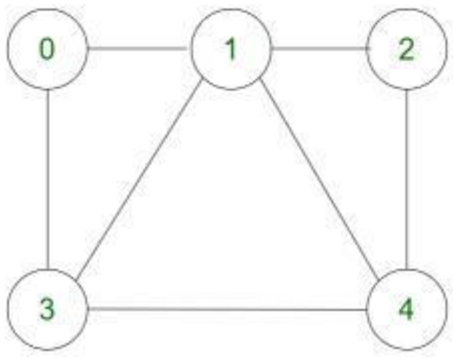
If graph contains a Hamiltonian cycle, it is called **Hamiltonian graph** otherwise it is **non-Hamiltonian**. Finding a Hamiltonian Cycle in a graph is a well-known NP Complete Problem, which means that there's no known efficient algorithm to solve it for all types of graphs. However, it can be solved for small or specific types of graphs.

The Hamiltonian Cycle problem has practical applications in various fields, such as **logistics, network design, and computer science**.

What is Hamiltonian Path?

Hamiltonian Path in a graph **G** is a path that visits every vertex of G exactly once and **Hamiltonian Path** doesn't have to return to the starting vertex. It's an open path.

Input: graph[][] = {{0, 1, 0, 1, 0},{1, 0, 1, 1, 1},{0, 1, 0, 0, 1},{1, 1, 0, 0, 1},{0, 1, 1, 1, 0}}



Input graph[][]

Output: {0, 1, 2, 4, 3, 0}.

Create an empty path array and add vertex **0** to it. Add other vertices, starting from the vertex **1**. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return **false**.

```
/* Java program for solution of Hamiltonian Cycle problem using backtracking */
class HamiltonianCycle
{       final int V = 5;
        int path[];
        boolean isSafe(int v, int graph[][], int path[], int pos)
        {
                //  Write Code Here
        .....   ......  .......
        }
        boolean hamCycleUtil(int graph[][], int path[], int pos)
        {       //  Write Code Here
        .....   ......  .......
                }


        /* A utility function to print solution */
        void printSolution(int path[])
        {
                System.out.println("Solution Exists: Following" + " is one Hamiltonian Cycle");
                for (int i = 0; i < V; i++)
                        System.out.print(" " + path[i] + " ");
                System.out.println(" " + path[0] + " ");
        }
        // driver program to test above function
        public static void main(String args[])
        {       //  Write Code Here
        .....   ......  .......
                hamiltonian.hamCycle(graph2);
        }
```

10. Backtracking

10.1 Partition of a set into K subsets with equal sum

Problem Statement and Example

Given an integer array of N elements, the task is to divide this array into K non-empty subsets such that the sum of elements in every subset is same. All elements of this array should be part of exactly one partition.

Input : arr = [2, 1, 4, 5, 6], K = 3

Output : Yes

we can divide above array into 3 parts with equal

sum as [[2, 4], [1, 5], [6]]

Input : arr = [2, 1, 5, 5, 6], K = 3

Output : No

It is not possible to divide above array into 3 parts with equal sum

Idea to Solve the Problem

We can solve this problem recursively, we keep an array for sum of each partition and a boolean array to check whether an element is already taken into some partition or not.

First we need to check some base cases,

If K is 1, then we already have our answer, complete array is only subset with same sum.

If N < K, then it is not possible to divide array into subsets with equal sum, because we can't divide the array into more than N parts.

If sum of array is not divisible by K, then it is not possible to divide the array. We will proceed only if k divides sum. Our goal reduces to divide array into K parts where sum of each part should be array_sum/K

```
// Java program to check whether an array can be partitioned into K subsets of equal sum
class GFG
{
   Static boolean isKPartitionPossibleRec(int arr[], int subsetSum[], boolean taken[],
                                          int subset, int K, int N, int curIdx, int limitIdx)
      {
            // Write Code Here
            ....          ....          ...
            }
            }
            return false;
}
```

```
// Method returns true if arr can be partitioned into K subsets with equal sum
static boolean isKPartitionPossible(int arr[], int N, int K)
{       // Write Code Here
        ....        ....        ...
}

// Driver code
public static void main(String[] args)
{
        int arr[] = {2, 1, 4, 5, 3, 3};
        int N = arr.length;
        int K = 3;

        if (isKPartitionPossible(arr, N, K))
                System.out.println("Partitions into equal sum is possible.");
        else
                System.out.println("Partitions into equal sum is not possible.");
}
}
```

10.2 Print all longest common sub-sequences in lexicographical order

Given two strings, the task is to print all the longest common sub-sequences in lexicographical order.

Examples:

Input : str1 = "abcabcaa", str2 = "acbacba"

Output: ababa

    abaca

    abcba

    acaba

    acaca

    acbaa

    acbca

This problem is an extension of longest common subsequence, We first find the length of LCS and store all LCS in a 2D table using Memoization (or Dynamic Programming). Then we search all characters from ‘a’ to ‘z’ (to output sorted order) in both strings. If a character is found in both strings and the current positions of the character lead to LCS, we recursively search all occurrences with current LCS length plus 1.

```
// Java program to find all LCS of two strings in  sorted order.
import java.io.*;
class GFG
{
```

```
static int MAX = 100;
static int lcslen = 0;
static int[][] dp = new int[MAX][MAX];
static int lcs(String str1, String str2, int len1, int len2, int i, int j)
{         // Write Code Here
                    ...        ....        ...
}
static void printAll(String str1, String str2, int len1, int len2,  char[] data, int indx1, int indx2,
                                                                                                   int currlcs)
{   // Write Code Here
                    ...        ....        ...
}

static void prinlAllLCSSorted(String str1, String str2)
{    // Write Code Here
                    ...        ....        ...
}

// Driver code
public static void main(String[] args)
{
          String str1 = "abcabcaa", str2 = "acbacba";
          prinlAllLCSSorted(str1, str2);
}
}
```

10.3 Print all Palindromic Partitions of a String using Bit Manipulation

Given a string, find all possible palindromic partitions of a given string.
Note that this problem is different from palindrome partitioning problem, there the task was to find the partitioning with minimum cuts in input string. Here we need to print all possible partitions.

Example:
Input: nitin
Output: n i t i n
n iti n
nitin

The idea is to consider the space between two characters in the string.

- If there **n** characters in the string, then there are **n-1** positions to put a space or say cut the string.
- Each of these positions can be given a binary number **1** (If a cut is made in this position) or **0** (If no cut is made in this position).
- This will give a total of **2n-1 partitions** and for each partition check whether this partition is palindrome or not.

```
import java.util.ArrayList;
import java.util.List;
class GFG {
        List<List<String>> ans = new ArrayList<>();
        boolean checkPalindrome(List<String> currPartition) {
                // Write Code Here
                ....        ...        ..
                }
                return true;
        }
        void generatePartition(String s, String bitString) {
                // Write Code Here
                ....        ...        ..
                }
        void bitManipulation(String s, String bitString) {
                // Write Code Here
                ....        ...        ..
        }

        public static void main(String[] args) {
                GFG ob = new GFG();
                List<List<String>> ans;
                // Write Code Here
                ....        ...        ..
}
```

10.4 Find shortest safe route in a path with landmines

Given a path in the form of a rectangular matrix having few landmines arbitrarily placed (marked as 0), calculate length of the shortest safe route possible from any cell in the first column to any cell in the last column of the matrix. We have to avoid landmines and their four adjacent cells (left, right, above and below) as they are also unsafe. We are allowed to move to only adjacent cells which are not landmines. i.e. the route cannot contains any diagonal moves.

Examples:

Input:
A 12 x 10 matrix with landmines marked as 0

[1 1 1 1 1 1 1 1 1 1]
[1 0 1 1 1 1 1 1 1 1]
[1 1 1 0 1 1 1 1 1 1]
[1 1 1 1 0 1 1 1 1 1]
[**1 1 1 1** 1 1 1 1 1 1]
[1 1 1 **1** 1 0 1 1 1 1]
[1 0 1 **1** 1 1 1 1 0 1]
[1 1 1 **1 1 1 1 1** 1 1]
[1 1 1 1 1 1 1 **1 1 1**]
[0 1 1 1 1 0 1 1 1 1]
[1 1 1 1 1 1 1 1 1 1]
[1 1 1 0 1 1 1 1 1 1]

Output:
Length of shortest safe route is 13 (Highlighted in **Bold**)

The idea is to use Backtracking. We first mark all adjacent cells of the landmines as unsafe. Then for each safe cell of first column of the matrix, we move forward in all allowed directions and recursively checks if they leads to the destination or not. If destination is found, we update the value of shortest path else if none of the above solutions work we return false from our function.

```
// Java program to find shortest safe Route in the matrix with landmines
import java.util.Arrays;
class GFG{
static final int R = 12;
static final int C = 10;
static int rowNum[] = { -1, 0, 0, 1 };
static int colNum[] = { 0, -1, 1, 0 };
static int min_dist;
// A function to check if a given cell (x, y) can be visited or not
static boolean isSafe(int[][] mat, boolean[][] visited, int x, int y)
{
            // Write Code Here
                        ….        ….        ….}
// A function to check if a given cell (x, y) is// a valid cell or not
static boolean isValid(int x, int y)
{
            // Write Code Here
                        ….        ….        ….}
static void markUnsafeCells(int[][] mat)
{             // Write Code Here
                        ….        ….        ….
}

static void findShortestPathUtil(int[][] mat, boolean[][] visited, int i, int j, int dist)
{          // Write Code Here
                        ….        ….        ….
}

// A wrapper function over findshortestPathUtil()
static void findShortestPath(int[][] mat)
{           // Write Code Here
                        ….        ….        ….
            }
// Driver code
public static void main(String[] args)
{           // Input matrix with landmines shown with number 0
          // Write Code Here
                        ….        ….        ….
            findShortestPath(mat);
}
}
```

10.5 Word Break Problem using Backtracking

Problem Statement
Given a valid sentence without any spaces between the words and a dictionary of valid English words, find all possible ways to break the sentence into individual dictionary words.
Example:
Consider the following dictionary
{ i, like, sam, sung, samsung, mobile, ice,
 and, cream, icecream, man, go, mango}

Input: "ilikesamsungmobile"
Output: i like sam sung mobile
 i like samsung mobile

Input: "ilikeicecreamandmango"
Output: i like ice cream and man go
 i like ice cream and mango
 i like icecream and man go

```
// print all possible partitions of a given string into dictionary words
import java.io.*;
import java.util.*;
class GFG {
// Prints all possible word breaks of given string
static void wordBreak(int n, List<String> dict, String s)
{
        String ans="";
        wordBreakUtil(n, s, dict, ans);
}

static void wordBreakUtil(int n, String s, List<String> dict, String ans)
{
        // Write code here
        ....    ....    ....
}
// main function
public static void main(String args[])
{       //Input str1 and str2
        // Write code here
        ....    ....    ....
        wordBreak(n1,dict,str1);
        System.out.println("\nSecond Test:");
        wordBreak(n2,dict,str2);
}
}
```

11. Branch and Bound

Branch and bound algorithms are used to find the optimal solution for combinatory, discrete, and general mathematical optimization problems.
A branch and bound algorithm provide an optimal solution to an NP-Hard problem by exploring the entire search space. Through the exploration of the entire search space, a branch and bound algorithm identify possible candidates for solutions step-by-step.
There are many optimization problems in computer science, many of which have a finite number of the feasible shortest path in a graph or minimum spanning tree that can be solved in polynomial time. Typically, these problems require a worst-case scenario of all possible permutations. The branch and bound algorithm create branches and bounds for the best solution.
Classification of Branch and Bound Problems:
The Branch and Bound method can be classified into three types based on the order in which the state space tree is searched.

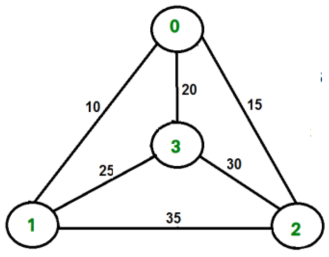
1. FIFO Branch and Bound

2. LIFO Branch and Bound

3. Least Cost-Branch and Bound

11.1 Travelling Salesperson Problem

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.scheduling algorithm is applied to schedule the jobs on a single processor to maximize the profits.



For example, consider the graph shown in figure on right side. A TSP tour in the graph is 0-1-3-2-0. The cost of the tour is 10+25+30+15 which is 80.
Branch and Bound Solution
As seen in the previous articles, in Branch and Bound method, for current node in tree, we compute a bound on best possible solution that we can get if we down this node. If the bound on best possible solution itself is worse than current best (best computed so far), then we ignore the subtree rooted with the node.
Note that the cost through a node includes two costs.
1) Cost of reaching the node from the root (When we reach a node, we have this cost computed)
2) Cost of reaching an answer from current node to a leaf (We compute a bound on this cost to decide whether to ignore subtree with this node or not).

- In cases of a **maximization problem**, an upper bound tells us the maximum possible solution if we follow the given node.

- In cases of a **minimization problem**, a lower bound tells us the minimum possible solution if we follow the given node.

In branch and bound, the challenging part is figuring out a way to compute a bound on best possible solution. Below is an idea used to compute bounds for Travelling salesman problem.
Cost of any tour can be written as below.

Cost of a tour T = (1/2) * ? (Sum of cost of two edges adjacent to u and in the tour T)
where u ? V
For every vertex u, if we consider two edges through it in T, and sum their costs. The overall sum for all vertices would be twice of cost of tour T (We have considered every edge twice.)

(Sum of two tour edges adjacent to u) >= (sum of minimum weight two edges adjacent to u)

Cost of any tour >= 1/2) * ? (Sum of cost of two minimum weight edges adjacent to u)
where u ? V
Now we have an idea about computation of lower bound. Let us see how to how to apply it state space search tree. We start enumerating all possible nodes (preferably in lexicographical order)

1. The Root Node: Without loss of generality, we assume we start at vertex "0" for which the lower bound has been calculated above.
Dealing with Level 2: The next level enumerates all possible vertices we can go to (keeping in mind that in any path a vertex has to occur only once) which are, 1, 2, 3… n (Note that the graph is complete). Consider we are calculating for vertex 1, Since we moved from 0 to 1, our tour has now included the edge 0-1. This allows us to make necessary changes in the lower bound of the root.

Lower Bound for vertex 1 =
Old lower bound - ((minimum edge cost of 0 + minimum edge cost of 1) / 2)
+ (edge cost 0-1)
How does it work? To include edge 0-1, we add the edge cost of 0-1, and subtract an edge weight such that the lower bound remains as tight as possible which would be the sum of the minimum edges of 0 and 1 divided by 2. Clearly, the edge subtracted can't be smaller than this.
Dealing with other levels: As we move on to the next level, we again enumerate all possible vertices. For the above case going further after 1, we check out for 2, 3, 4, …n.
Consider lower bound for 2 as we moved from 1 to 1, we include the edge 1-2 to the tour and alter the new lower bound for this node.
Lower bound(2) = Old lower bound - ((second minimum edge cost of 1 + minimum edge cost of 2)/2)
+ edge cost 1-2)
Note: The only change in the formula is that this time we have included second minimum edge cost for 1, because the minimum edge cost has already been subtracted in previous level.

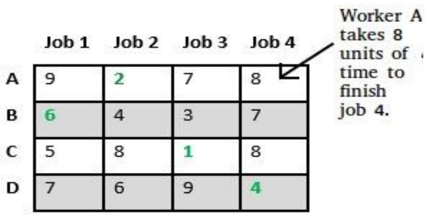
```
// Java program to solve Traveling Salesman Problem using Branch and Bound.
import java.util.*;
class GFG
{	static int N = 4;
	static int final_path[] = new int[N + 1];
	static boolean visited[] = new boolean[N];
	static int final_res = Integer.MAX_VALUE;
	static void copyToFinal(int curr_path[])
	{   // Write Code Here
```

```
                ...         ....        ....
        }
        // Function to find the minimum edge cost having an end at the vertex i
        static int firstMin(int adj[][], int i)
        {
            // Write Code Here
                ...         ....        ....}
        static int secondMin(int adj[][], int i)
        {
                // Write Code Here
                ...         ....        ....
        }
        static void TSPRec(int adj[][], int curr_bound, int curr_weight, int level, int curr_path[])
        {       // Write Code Here
                ...         ....        ....        }

        static void TSP(int adj[][])
        {
                // Write Code Here
                ...         ....        ....        }

        // Driver code
        public static void main(String[] args)
        {
                //Adjacency matrix for the given graph
                // Write Code Here
                ...         ....        ....
                TSP(adj);
                System.out.printf("Minimum cost : %d\n", final_res);
                System.out.printf("Path Taken : ");
                for (int i = 0; i <= N; i++)
                {
                        System.out.printf("%d ", final_path[i]);
                }
        }
}
```

11.2 Job Assignment Problem

Problem Statement

Let there be N workers and N jobs. Any worker can be assigned to perform any job, incurring some cost that may vary depending on the work-job assignment. It is required to perform all jobs by assigning exactly one worker to each job and exactly one job to each agent in such a way that the total cost of the assignment is minimized.



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Worker A takes 8 units of time to finish job 4.

An example job assignment problem. Green values show optimal job assignment that is A-Job2, B-Job1 C-Job3 and D-Job4

Finding Optimal Solution using Branch and Bound

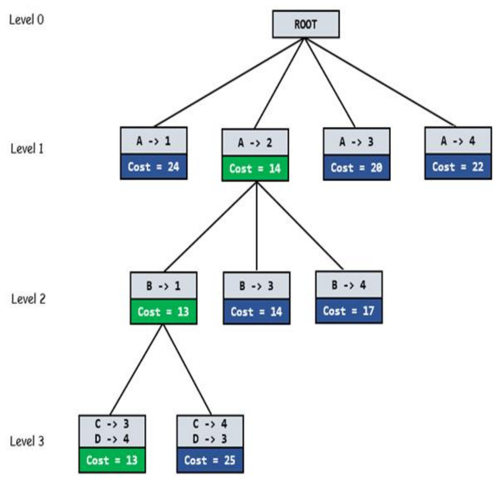
The selection rule for the next node in BFS and DFS is "blind". i.e. the selection rule does not give any preference to a node that has a very good chance of getting the search to an answer node quickly. The search for an optimal solution can often be speeded by using an "intelligent" ranking function, also called an approximate cost function to avoid searching in sub-trees that do not contain an optimal solution. It is similar to BFS-like search but with one major optimization. Instead of following FIFO order, we choose a live node with least cost. We may not get optimal solution by following node with least promising cost, but it will provide very good chance of getting the search to an answer node quickly.

There are two approaches to calculate the cost function:

1. For each worker, we choose job with minimum cost from list of unassigned jobs (take minimum entry from each row).

2. For each job, we choose a worker with lowest cost for that job from list of unassigned workers (take minimum entry from each column).

Below diagram shows complete search space diagram showing optimal solution path in green.



Complete Algorithm:

```
/* findMinCost uses Least() and Add() to maintain the list of live nodes
   Least() finds a live node with least cost, deletes  it from the list and returns it
   Add(x) calculates cost of x and adds it to the list of live nodes
   Implements list of live nodes as a min heap */
```

```
// Search Space Tree Node node
{
  int job_number;
  int worker_number;
  node parent;
  int cost;
}
// Input: Cost Matrix of Job Assignment problem Output: Optimal cost and Assignment of Jobs
algorithm findMinCost (costMatrix mat[][])
{
  // Initialize list of live nodes(min-Heap) with root of search tree i.e. a Dummy node
  while (true)
  {
    // Find a live node with least estimated cost
    E = Least();
    // The found node is deleted from the list of live nodes
    if (E is a leaf node)
    {
      printSolution();
      return;
    }

   for each child x of E
   {
      Add(x); // Add x to list of live nodes;
      x->parent = E; // Pointer for path to root
   }
  }
}
```

```java
import java.util.*;
// Node class represents a job assignment
class Node {
    Node parent; // parent node
    int pathCost; // cost to reach this node
    int cost; // lower bound cost
    int workerID; // worker ID
    int jobID; // job ID
    boolean assigned[]; // keeps track of assigned jobs
    public Node(int N) {
        assigned = new boolean[N]; // initialize assigned jobs array
    }
}
public class Main {
    static final int N = 4; // number of workers and jobs

    // Function to create a new search tree node
    static Node newNode(int x, int y, boolean assigned[], Node parent) {
        Node node = new Node(N);
        for (int j = 0; j < N; j++)
```

```
            node.assigned[j] = assigned[j];
        if (y != -1) {
            node.assigned[y] = true;
        }
        node.parent = parent;
        node.workerID = x;
        node.jobID = y;
        return node;
    }
   static int calculateCost(int costMatrix[][], int x, int y, boolean assigned[]) {
        // Write Code Here
            ....        .....        ...    }
    // Function to print job assignment
    static void printAssignments(Node min) {
        // Write Code Here
            ....        .....        ...    }
    // Function to solve Job Assignment Problem using Branch and Bound
    static int findMinCost(int costMatrix[][]) {
// Write Code Here
            ....        .....        ...    }

    public static void main(String[] args) {
        // Write Code Here for input
            ....        .....        ...
        System.out.println("\nOptimal Cost is " + findMinCost(costMatrix));
    }
}
```

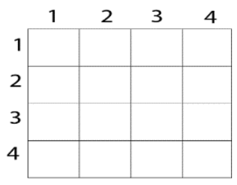
11.3 N-Queen Problem Using Branch and Bound

N-Queens Problem Statement:

N - Queens problem is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

It can be seen that for n =1, the problem has a trivial solution, and no solution exists for n =2 and n =3. So first we will consider the 4 queens problem and then generate it to n - queens problem.

Given a 4 x 4 chessboard and number the rows and column of the chessboard 1 through 4.



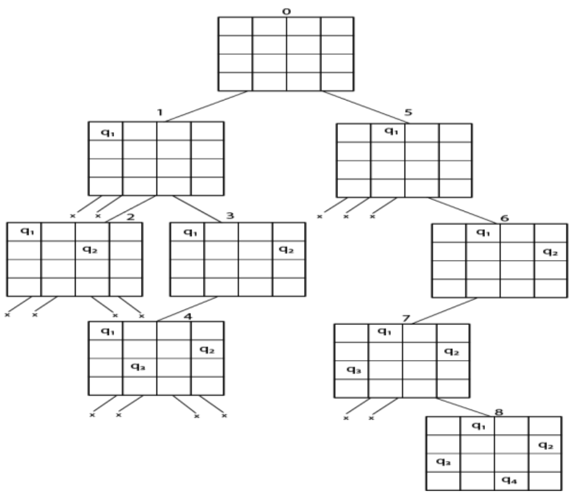
4x4 chessboard

Since, we have to place 4 queens such as q_1 q_2 q_3 and q_4 on the chessboard, such that no two queens attack each other. In such a conditional each queen must be placed on a different row, i.e., we put queen "i" on row "i."

Now, we place queen q_1 in the very first acceptable position (1, 1). Next, we put queen q_2 so that both these queens do not attack each other. We find that if we place q_2 in column 1 and 2, then the dead end is encountered. Thus the first acceptable position for q_2 in column 3, i.e. (2, 3) but then no position is left for placing queen 'q_3' safely. So we backtrack one step and place the queen 'q_2' in (2, 4), the next best possible solution. Then we obtain the position for placing 'q_3' which is (3, 2). But later this position also leads to a dead end, and no place is found where 'q_4' can be placed safely. Then we have to backtrack till 'q_1' and place it to (1, 2) and then all other queens are placed safely by moving q_2 to (2, 4), q_3 to (3, 1) and q_4 to (4, 3). That is, we get the solution (2, 4, 1, 3). This is one possible solution for the 4-queens problem. For another possible solution, the whole method is repeated for all partial solutions. The other solutions for 4 - queens problems is (3, 1, 4, 2) i.e.

	1	2	3	4
1			q_1	
2	q_2			
3				q_3
4		q_4		

The implicit tree for 4 - queen problem for a solution (2, 4, 1, 3) is as follows:



Place (k, i) returns a Boolean value that is true if the kth queen can be placed in column i. It tests both whether i is distinct from all previous costs $x_1, x_2, \ldots x_{k-1}$ and whether there is no other queen on the same diagonal.

Using place, we give a precise solution to then n- queens problem.

```
Place (k, i)
  {
    For j ← 1 to k - 1
     do if (x [j] = i)
       or (Abs x [j]) - i) = (Abs (j - k))
    then return false;
     return true;
}
```

Place (k, i) return true if a queen can be placed in the kth row and ith column otherwise return is false.

x [] is a global array whose final k - 1 values have been set. Abs (r) returns the absolute value of r.

```
N - Queens (k, n)
```

```
{
   For i ← 1 to n
      do if Place (k, i) then
   {
      x [k] ← i;
      if (k ==n) then
        write (x [1....n));
      else
      N - Queens (k + 1, n);
   }
}
```

```
// Java program to solve N Queen Problem  using Branch and Bound
import java.io.*;
import java.util.Arrays;
class GFG {
static int N = 8;
static void printSolution(int board[][])
{
        //  Code Here
                ...        ...        .... }

// A Optimized function to check if a queen  can be placed on board[row][col]
static boolean isSafe(int row, int col,  ....
//  Code Here
                ...        ...        ...}
// A recursive utility function to solve N Queen problem
static boolean solveNQueensUtil( .......
        { //  Code Here
                ...        ...        ... }
static boolean solveNQueens()
{
        //  Code Here
                ...        ...        ...
}
// Driver code
public static void main(String[] args)
{       solveNQueens();
}
}
```

11.4 0/1 Knapsack Problem

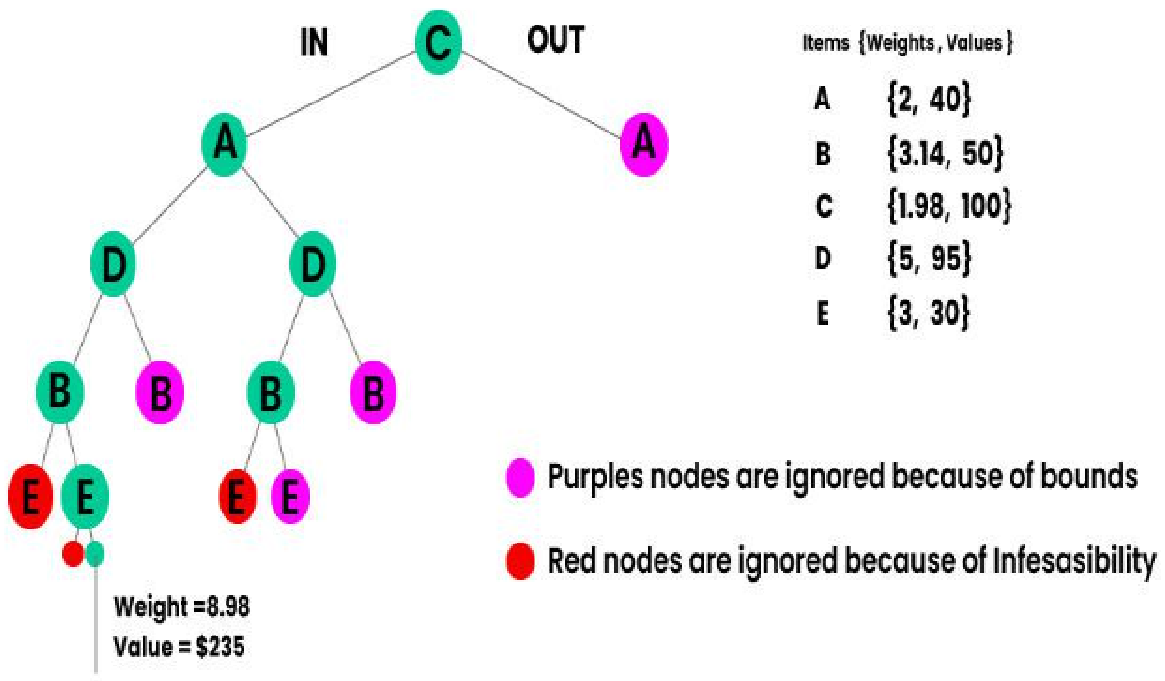
Problem Statement:

Given two arrays **v[]** and **w[]** that represent values and weights associated with n items respectively. Find out the maximum value subset(Maximum Profit) of **v[]** such that the sum of the weights of this subset is smaller than or equal to Knapsack capacity **Cap(W)**.

Note: The constraint here is we can either put an item completely into the bag or cannot put it at all. It is not possible to put a part of an item into the bag.

Implementation of 0/1 Knapsack using Branch and Bound:

We will solve this problem using Branch and Bound technique.



Implementation of 0/1 Knapsack using Branch and Bound

Approach to find bound for every node for 0/1 Knapsack

The idea is to use the fact that the Greedy algorithm provides the best solution for Fractional Knapsack problem. To check if a particular node can give us a better solution or not, we compute the optimal solution (through the node) using Greedy approach. If the solution computed by Greedy approach itself is more than the best so far, then we can't get a better solution through the node.

Algorithm:

- Sort all items in decreasing order of ratio of value per unit weight so that an upper bound can be computed using Greedy Approach.
- Initialize maximum profit, maxProfit = 0
- Create an empty queue, Q.
- Create a dummy node of decision tree and enqueue it to Q. Profit and weight of dummy node are 0.
- Do following while Q is not empty.

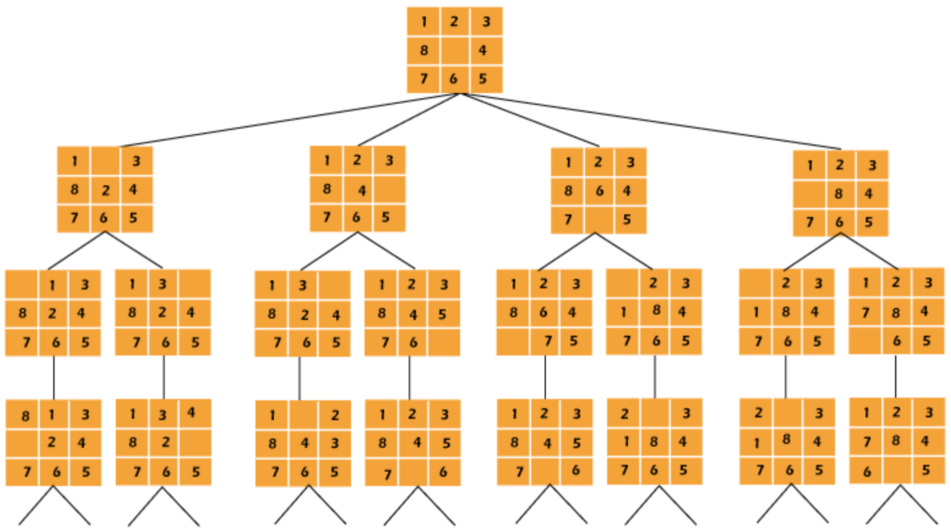
- Extract an item from Q. Let the extracted item be u.
- Compute profit of next level node. If the profit is more than maxProfit, then update maxProfit.
- Compute bound of next level node. If bound is more than maxProfit, then add next level node to Q.
- Consider the case when next level node is not considered as part of solution and add a node to queue with level as next, but weight and profit without considering next level nodes.

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.PriorityQueue;
class Item {
      float weight;
      int value;
      Item(float weight, int value) {
            this.weight = weight;
            this.value = value;
      }
}
class Node {
      int level, profit, bound;
      float weight;

      Node(int level, int profit, float weight) {
            this.level = level;
            this.profit = profit;
            this.weight = weight;
      }
}

public class KnapsackBranchAndBound {
      // Write Code Here
      ….    …..   ….    };
      static int bound(Node u, int n, int W, Item[] arr) {
            // Write Code Here
      ….    …..   ….    }

      static int knapsack(int W, Item[] arr, int n) {
            // Write Code Here
      ….    …..   ….    }

      public static void main(String[] args) {
            int W = 10;
            Item[] arr = {
                new Item(2, 40),
                new Item(3.14f, 50),
                new Item(1.98f, 100),
                new Item(5, 95),
                new Item(3, 30)
        };
        int n = arr.length;
        int maxProfit = knapsack(W, arr, n);
        System.out.println("Maximum possible profit = " +
maxProfit);
    }
}
```

11.5 8 Puzzle Problem

The 8 puzzle problem solution is covered in this article. A **3 by 3 board with 8 tiles** (each tile has a number from 1 to 8) and a **single empty space** is provided. The goal is to **use the vacant space to arrange the numbers on the tiles** such that they match the final arrangement. **Four neighbouring** (left, right, above, and below) **tiles** can be moved into the available area.

For instance,



By avoiding searching in sub-trees which do not include an answer node, an **"intelligent" ranking function**, also known as an **approximation costs function**, may frequently **speed up the search for an answer node**.

Basically, **Branch and Bound** involves three different kinds of nodes.

A **live node** is a created node whose children have not yet been formed.

The offspring of the **E-node** which is a live node, are now being investigated. Or to put it another way, an E-node is a node that is currently expanding.

A created node which is not to be developed or examined further is referred to as a **dead node**. A dead node has already extended all of its offspring.

Cost function:

In the search tree, each **node Y** has a corresponding cost. The next **E-node** may be found using the **cost function**. The E-node with the **lowest costs** is the next one. The function can be defined as:

C(Y) = g(Y) + h(Y) Where

g(Y) = the costs of reaching to the current node **from** the root.

h(Y) = the costs of reaching to an answer node **from** the Y.

The optimum costs function for an algorithm for 8 puzzles is :

We suppose that it will costs **one unit** to move a tile in any direction. In light of this, we create the following costs function for the 8-puzzle algorithm:

c(y) = f(y) + h(y) where

f(y) = the path's total length **from** the root y.

h(y) = the amount of the non-blank tiles which are **not in** their final goal position (misplaced tiles).

To change state y into a desired state, there are at least h(y) movements required.

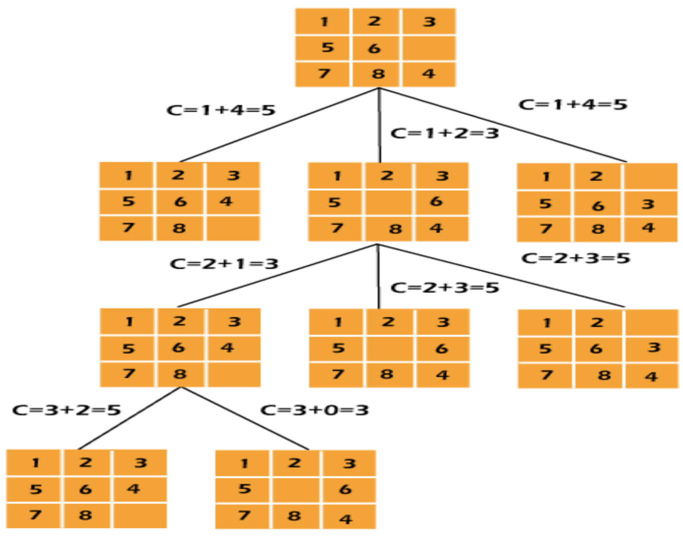
There is an algorithm for **estimating the unknown value of h(y),** which is accessible.

Final algorithm:

In order to maintain **the list of live nodes**, algorithm **LCSearch** employs the functions **Least()** and **Add().**

Least() identifies a live node with the **least c(y),** removes it from the list, and returns it.
Add(y) adds **y** to the list of live nodes.
Add(y) implements the list of live nodes as a **min-heap**.

The route taken by the aforementioned algorithm to arrive at the final configuration of the 8-Puzzle from the starting configuration supplied is shown in the **diagram below**. Keep in mind that only **nodes with the lowest costs function** value are extended.



```
// Java Program to print path from root node to destination node for N*N -1 puzzle
// algorithm using Branch and Bound The solution assumes that instance of puzzle is olvable
import java.io.*;
import java.util.*;
class GFG
{       public static int N = 3;
        public static class Node
        { // Code Here
        ...     ...     ...}
        // Function to print N x N matrix
        public static void printMatrix(int mat[][]){
                // Code Here
        ...     ...     ...}
        // Function to allocate a new node
        public static Node newNode(int mat[][], int x, int y,....
                // Code Here
        ...     ...     ... }

        public static int calculateCost(int initialMat[][], int finalMat[][])
        {
                // Code Here
        ...     ...     ...     }

        // Function to check if (x, y) is a valid matrix coordinate
        public static int isSafe(int x, int y)
        // Code Here
        ...     ...     ...}

        // Comparison object to be used to order the heap
        public static class comp implements Comparator<Node>{....

        public static void solve(int initialMat[][], int x, int y, int finalMat[][])
        {
```

```
        // Code Here
        ...       ...       ...       }

        //Driver Code
        public static void main (String[] args)
        {
                // Code Here for input
        ...       ...       ...

                solve(initialMat, x, y, finalMat);
        }
}
```

12. Branch and Bound

12.1 Generate Binary Strings of length N using Branch and Bound

Problem Statement and Example

The task is to generate a binary string of length N using branch and bound technique

Examples:

Input: N = 3

Output: 000 001 010 011 100 101 110 111

Explanation: Numbers with 3 binary digits are 0, 1, 2, 3, 4, 5, 6, 7

Input: N = 2

Output: 00 01 10 11

Approach: Generate Combinations using Branch and Bound :

- It starts with an empty solution vector .
- While Queue is not empty remove partial vector from queue.
- If it is a final vector print the combination else,
- For the next component of partial vector create k child vectors by fixing all possible states for the next component insert vectors into the queue.

```
// Java Program to generate Binary Strings using Branch and Bound
import java.io.*;
import java.util.*;
// Creating a Node class
class Node {
        int soln[];
        int level;
        ArrayList<Node> child;
        Node parent;
        Node(Node parent, int level, int N)
        {
                this.parent = parent;
                this.level = level;
                this.soln = new int[N];
        }
}

class GFG {
          // Write Code Here
        ...      ...      ...      }

        // Driver code
        public static void main(String args[])
        {
                N = 3;
                Node root = new Node(null, 0, N);
                Q = new LinkedList<Node>();
                Q.add(root);
                while (Q.size() != 0) {
                        Node E = Q.poll();
                        generate(E);
                }
        }
}
```

12.2 Feature selection using branch and bound algorithm

A Feature selection is critical in the domains of machine learning and data analysis since it assists in identifying the most essential and informative features in a dataset. It is a procedure that seeks to extract relevant features that will help with analysis and modelling jobs. The branch and bound method is an effective feature selection tool. −

As the volume of data grows at an exponential rate, it is becoming increasingly vital to build efficient algorithms capable of quickly identifying the ideal subset of attributes. In this post, we will look at feature selection and how the branch and bound method may be used to improve the efficiency and accuracy of the feature selection process.

What is Feature Selection?

In machine learning and statistics, feature selection refers to the process of choosing a subset of relevant features that are most informative for a given task. By selecting the right features, we aim to improve the model's performance, reduce computational complexity, and mitigate the risk of overfitting.

Significance of Feature Selection

Feature selection offers several advantages in the field of data analysis and machine learning −

- **Improved Model Performance** − By selecting the most relevant features, we can enhance the accuracy and predictive power of the model. Irrelevant or redundant features can introduce noise and hinder model performance.
- **Reduced Dimensionality** − Feature selection helps in reducing the number of dimensions or attributes in the dataset. This reduction simplifies the problem space, improves computational efficiency, and facilitates better model interpretability.
- **Elimination of Overfitting** − Including irrelevant features in the model can lead to overfitting, where the model becomes too specific to the training data and fails to generalize well on unseen data. Feature selection mitigates this risk by focusing on the most informative features.
- **Faster Training and Inference** − By reducing the dimensionality of the dataset, feature selection can significantly speed up the training and inference phases of the model. This is particularly important when dealing with large-scale datasets.

What is Branch and Bound Algorithm?

The Branch and Bound algorithm is a systematic approach to finding the optimal subset of features by exploring all possible feature combinations. It utilizes a divide-and-conquer strategy coupled with intelligent pruning to efficiently search the feature space. The algorithm begins with an initial bound and progressively explores different branches to narrow down the search space until the optimal subset is found.

Algorithm

Step 1: Initialization

The Branch and Bound algorithm starts by initializing the search process. This involves setting up the initial bounds, creating a priority queue to track the best feature subsets, and defining other necessary data structures.

Step 2: Generate Initial Bounds

To guide the search process, the algorithm generates initial bounds based on the evaluation criteria. These bounds provide an estimate of the best possible solution and help in pruning unpromising branches.

Step 3: Explore Branches

The algorithm explores different branches or paths in the search tree. Each branch represents a subset of features. It evaluates the quality of each branch based on a predefined evaluation metric and decides whether to further explore or prune the branch.

Step 4: Update Bounds

As the algorithm progresses and explores different branches, it updates the bounds dynamically. This allows for more accurate pruning decisions and helps in speeding up the search process.

Step 5: Pruning and Stopping Criteria

Branch and Bound employ pruning techniques to eliminate branches that are guaranteed to be suboptimal. This reduces the search space and focuses on more promising feature subsets. The algorithm continues the search until a stopping criterion is met, such as finding the optimal subset or reaching a predefined computational limit.

Example Demonstration

Let's consider a simple example to illustrate the working of the Branch and Bound algorithm. Suppose we have a dataset with 10 features, and we want to find the optimal subset of features for a classification task. The algorithm would systematically explore different feature combinations, evaluate their performance, and prune unpromising branches until it discovers the subset with the highest evaluation metrics, such as accuracy or information gain.

Example

Below is the program for the above example –

```
import itertools
def evaluate_subset(subset):
    return len(subset)
def branch_and_bound(features, k):
  n = len(features)
  best_subset = []
  best_score = 0.0
  def evaluate_branch(subset):
    Write Code Here
        ..................
  def backtrack(subset, idx):
Write Code Here
        ..................
return best_subset
# Example usage
if __name__ == '__main__':
   features = ['Feature A', 'Feature B', 'Feature C', 'FeatureD', 'Feature E', 'Feature F', 'Feature G', 'Feature H', 'Feature I', 'Feature J']
  k = 3  # Number of features to select
  selected_features = branch_and_bound(features, k)
  print(f"Selected Features: {selected_features}")
```

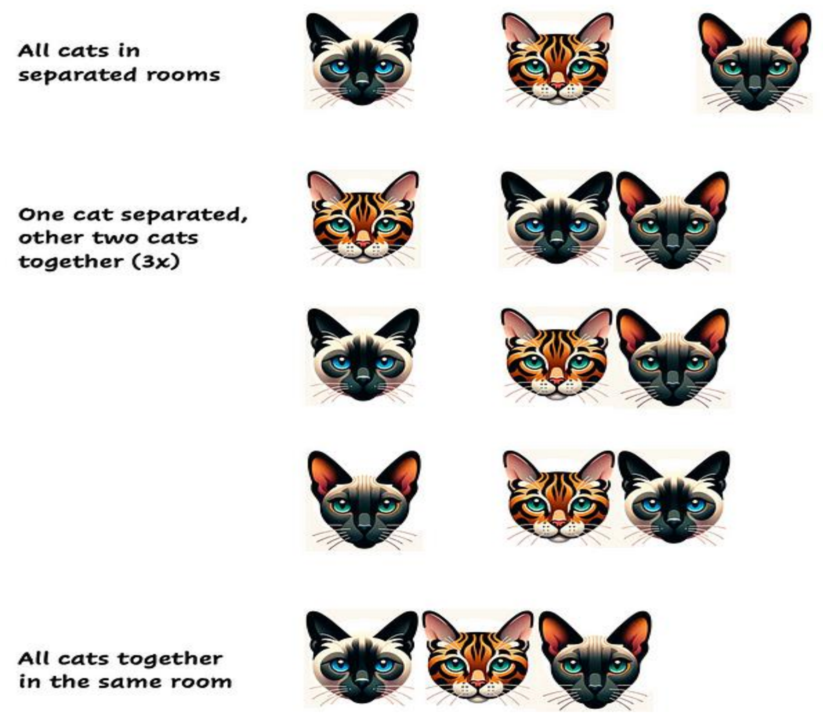
12.3 Mixed Integer Programming (MIP) using Branch and bound

Problem Statement:

Imagine you are the owner of a cat shelter. Every day, pet owners can bring their cats and you take care of them. Many people adopted a cat during COVID, but now everyone needs to be back at the office. Because of this your company is doing great.

Actually, a bit too great. You are having difficulties with placing all the cats in the rooms of your building. Sometimes you need to decline people because there are too many requests. That is why you decided to create an optimization algorithm to help you find the lowest number of rooms possible for all the cat registrations.

Let’s take a look at an example. Imagine that 3 cats requested to stay at your shelter. Their names are Lily, Charlie, and Meowster. How can we divide these three cats in different rooms? We need at most three rooms, and here are all the possible solutions of grouping the cats:


All cats in separated rooms
One cat separated, other two cats together (3x)
All cats together in the same room

Partitions of cats

Partitions and the Bell number

As you can see, there are 5 possible ways to group the 3 cats. In math, the name for one way of grouping elements of a set is a *partition*. The *Bell number* corresponds to the total number of possible partitions for a given set (in our case, with the three cats we can create 5 partitions). It's from the field of combinatorics.

The recursive formula for calculating the next Bell number looks like this:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

Recursive formula for calculating the Bell number. Image by author.

This number increases rapidly:

```
import org.apache.commons.math3.optim.linear.LinearObjectiveFunction;
import org.apache.commons.math3.optim.linear.Relationship;
import org.apache.commons.math3.optim.linear.LinearConstraint;
import org.apache.commons.math3.optim.linear.NoFeasibleSolutionException;
import org.apache.commons.math3.optim.linear.SimplexSolver;
import org.apache.commons.math3.optim.linear.NonNegativeConstraint;
import org.apache.commons.math3.optim.linear.UnboundedSolutionException;
import org.apache.commons.math3.optim.MaxIter;
import org.apache.commons.math3.optim.OptimizationData;
import java.util.ArrayList;
import java.util.List;
public class MIPSolver {
    public static void main(String[] args) {
        // Write Code Here
                    ....        ....        ...
        };

        try {
            SimplexSolver solver = new SimplexSolver();
            double[] solution = solver.optimize(f, constraints, optData).getPoint();
            System.out.println("Optimal solution: x1 = " + solution[0] + ", x2 = " + solution[1]);
        } catch (NoFeasibleSolutionException e) {
            System.out.println("No feasible solution found.");
        } catch (UnboundedSolutionException e) {
            System.out.println("Solution is unbounded.");
        }
    }
}
```

Solving the Cat Problem
With this knowledge, let's get back to our initial problem. Let's code branch and bound in Python. We will use breadth first search, but feel free to reuse this code to try other exploration strategies.
Some rules we will implement:

- The score of a solution will be equal to the number of rooms we use. Obviously we want to minimize this.
- Feasibility: A room can never have more than 5 cats, and the total weight of the cats in a room should not exceed 25 kilograms. Also, a room can have at most one cat with an 'angry' character, otherwise we will have cat fights...
- The lower bound of a node is calculated by the current score of the node plus the minimum number of extra rooms we need based on the number of angry cats.

Let's start with a Cat class.

```
import random
from typing import List
import numpy as np
class Cat:
    """
    A cat has a name, a weight and a character.
    """

    def __init__(self, name: str, weight: int, character: str):
        self.name = name
        self.weight = weight
        self.character = character


def generate_n_cats(n: int) -> List[Cat]:
    """
    Generate n cats with random names, weights and characters.
    """
    cats = []
    for i in range(n):
        name = f"cat_{i}"
        weight = random.randint(1, 10)
        character = np.random.choice(["sweet", "angry"], p=[0.8, 0.2])
        cats.append(Cat(name, weight, character))
    return cats
```

With the generate_n_cats function, we can generate as many cats as we like.
In the next code snippet, we will code the branch and bound algorithm:

```
import time
from typing import List
from cat_generator import Cat, generate_n_cats
class Node:
    def __init__(self, partition, remaining_cats, score):
        self.partition = partition
        self.remaining_cats = remaining_cats
        self.score = score

    def is_feasible(self):
            Write Code Here
            ...........
       return True
    def calculate_lower_bound(self):
        Write Code Here
              ..........................
        return lower_bound

    @staticmethod
    def calculate_weight(group):
        """
        Calculate the weight of a group.
        """
        return sum([cat.weight for cat in group])

    @staticmethod
    def count_angry_cats(group):
        """
        Check how many angry cats are in a group.
        """
        angry_cats = [cat for cat in group if cat.character == "angry"]
        return len(angry_cats)
class BranchAndBound:
    def __init__(self, cats: List[Cat]) -> None:
        self.cats = cats
        self.solution_type = "Branch and Bound"

    def create_solution(self, timelimit: int = 300):
            Write Code Here
            ...................
        return best_partition

    def score(self, partition) -> int:
        """
        Calculate the score of a partition.
        """
        return len(partition)
if __name__ == "__main__":
    cats = generate_n_cats(15)
    builder = BranchAndBound(cats)
    solution = builder.create_solution()
    print(f"Score of {builder.solution_type}: {builder.score(solution)}")
    print(f"Time of {builder.solution_type}: {builder.time}")
    for group in solution:
        print(group)
    print(builder.score(solution))
```

12.4 Generate Binary Strings of length N using Branch and Bound

The task is to generate a binary string of length N using branch and bound technique **Examples:**

Input: N = 3 **Output:** 000 001 010 011 100 101 110 111 **Explanation:** Numbers with 3 binary digits are 0, 1, 2, 3, 4, 5, 6, 7 **Input:** N = 2 **Output:** 00 01 10 11
Approach: Generate Combinations using Branch and Bound:

- It starts with an empty solution vector.
- While Queue is not empty remove partial vector from queue.
- If it is a final vector print the combination else,
- For the next component of partial vector create k child vectors by fixing all possible states for the next component insert vectors into the queue.

Below is the implementation of the above approach

```
// Java Program to generate Binary Strings using Branch and Bound
import java.io.*;
import java.util.*;
// Creating a Node class
class Node {
        int soln[];
        int level;
        ArrayList<Node> child;
        Node parent;
        Node(Node parent, int level, int N)
        {
                this.parent = parent;
```

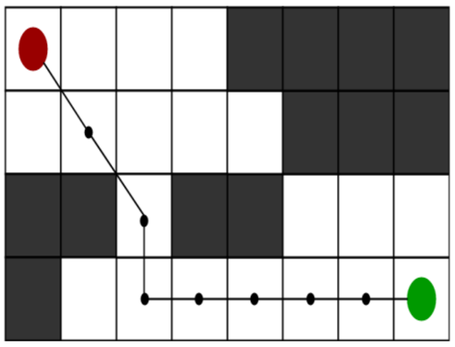
```
                    this.level = level;
                    this.soln = new int[N];
          }
}

class GFG {
           // Write Code Here
          ...          ...          ...          }

          // Driver code
          public static void main(String args[])
          {
                    N = 3;
                    Node root = new Node(null, 0, N);
                    Q = new LinkedList<Node>();
                    Q.add(root);
                    while (Q.size() != 0) {
                              Node E = Q.poll();
                              generate(E);
                    }
          }
}
```

12.5 A* Search Algorithm

Motivation

To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

We can consider a 2D Grid having several obstacles and we start from a source cell (colored red below) to reach towards a goal cell (colored green below)



What is A* Search Algorithm?

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Why A* Search Algorithm?

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.

And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

Explanation

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.

What A* Search Algorithm does is that at each step it picks the node according to a value-‘**f**’ which is a parameter equal to the sum of two other parameters – ‘**g**’ and ‘**h**’. At each step it picks the node/cell having the lowest ‘**f**’, and process that node/cell.

We define ‘**g**’ and ‘**h**’ as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.

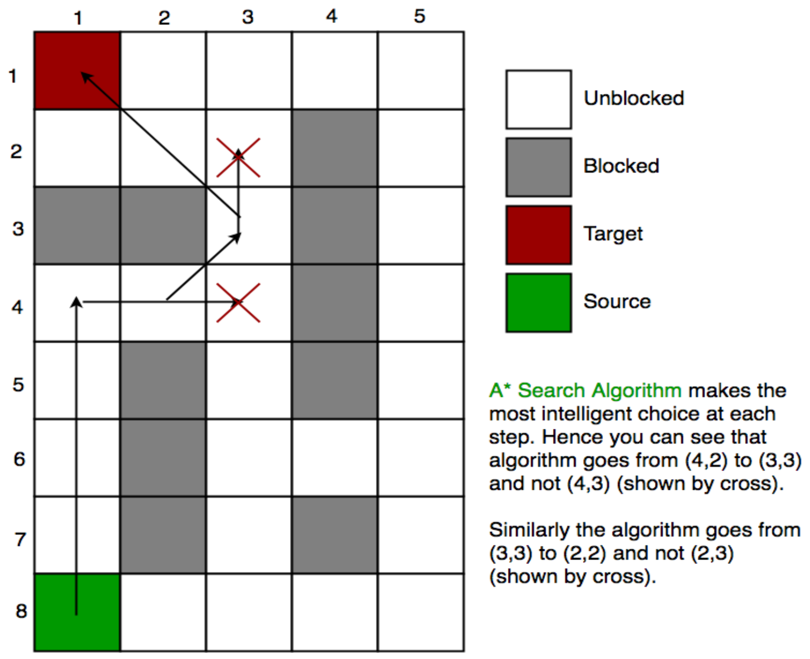
h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.

Algorithm

We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

```
// A* Search Algorithm
1.  Initialize the open list
2.  Initialize the closed list
    put the starting node on the open
    list (you can leave its f at zero)
3.  while the open list is not empty
    a) find the node with the least f on
       the open list, call it "q"
    b) pop q off the open list

    c) generate q's 8 successors and set their
       parents to q

    d) for each successor
        i) if successor is the goal, stop search

        ii) else, compute both g and h for successor
          successor.g = q.g + distance between
                              successor and q
          successor.h = distance from goal to
          successor (This can be done using many
          ways, we will discuss three heuristics-
          Manhattan, Diagonal and Euclidean
          Heuristics)
          successor.f = successor.g + successor.h
        iii) if a node with the same position as
            successor is in the OPEN list which has a
           lower f than successor, skip this successor
        iV) if a node with the same position as
            successor  is in the CLOSED list which has
            a lower f than successor, skip this successor
            otherwise, add  the node to the open list
    end (for loop)

    e) push q on the closed list
    end (while loop)
```

So suppose as in the below figure if we want to reach the target cell from the source cell, then the A* Search algorithm would follow path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristics.



1
2
3
4
5
1
2
3
4
5
6
7
8
Unblocked
Blocked
Target
Source
A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).
Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Heuristics

We can calculate **g** but how to calculate h ?

We can do things.

A) Either calculate the exact value of h (which is certainly time consuming).

OR

B) Approximate the value of h using some heuristics (less time consuming).

We will discuss both of the methods.

A) **Exact Heuristics** –

We can find exact values of h, but that is generally very time consuming.

Below are some of the methods to calculate the exact value of h.

1) Pre-compute the distance between each pair of cells before running the A* Search Algorithm.

2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance

B) Approximation Heuristics –

There are generally three approximation heuristics to calculate h –

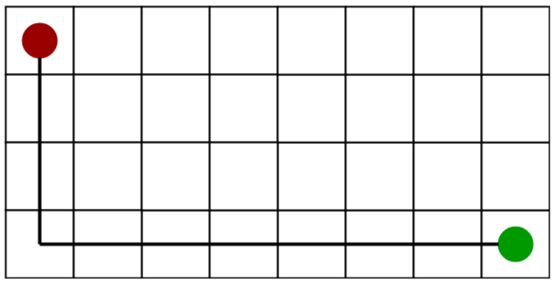
1) Manhattan Distance –

- It is nothing but the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

h = abs (current_cell.x – goal.x) +
abs (current_cell.y – goal.y)

- When to use this heuristic? – When we are allowed to move only in four directions only (right, left, top, bottom)

The Manhattan Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



2) Diagonal Distance-

- It is nothing but the maximum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

dx = abs(current_cell.x – goal.x)

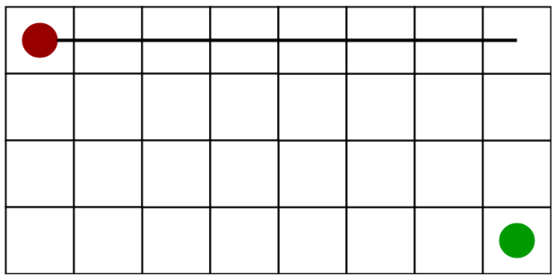
dy = abs(current_cell.y – goal.y)

h = D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)

where D is length of each node(usually = 1) and D2 is diagonal distance between each node (usually = sqrt(2)).

- When to use this heuristic? – When we are allowed to move in eight directions only (similar to a move of a King in Chess)

The Diagonal Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



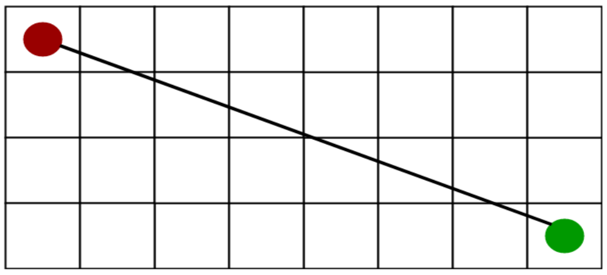
3) Euclidean Distance-

- As it is clear from its name, it is nothing but the distance between the current cell and the goal cell using the distance formula

h = sqrt ((current_cell.x – goal.x)2 +
(current_cell.y – goal.y)2)

- When to use this heuristic? – When we are allowed to move in any directions.

The Euclidean Distance Heuristics is shown by the below figure (assume red spot as source cell and green spot as target cell).



Relation (Similarity and Differences) with other algorithms-

Dijkstra is a special case of A* Search Algorithm, where h = 0 for all nodes.

Implementation

We can use any data structure to implement open list and closed list but for best performance, we use a **set** data structure of C++ STL(implemented as Red-Black Tree) and a boolean hash table for a closed list.

The implementations are similar to Dijkstra's algorithm. If we use a Fibonacci heap to implement the open list instead of a binary heap/self-balancing tree, then the performance will become better (as Fibonacci heap takes O(1) average time to insert into open list and to decrease key)

```
import java.util.*;
class Node {
   int x, y;
   int g, h;
   Node parent;
   public Node(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getF() {
        return g + h;
    }
}

public class AStar {
    public static final int[][] GRID = {
            // Write Input Here
                    …           ….          ….    };
    public static final int[][] DIRS = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    public static int heuristic(int x, int y, int tx, int ty) {
        return Math.abs(tx - x) + Math.abs(ty - y);
    }
    public static List<Node> astar(int[][] grid, int sx, int sy, int tx, int ty) {
        // Write code Here
            ….          ….          ..          }
        return null; // No path found
    }

    public static void main(String[] args) {
        int startX = 0, startY = 0;
        int targetX = 4, targetY = 4;

        List<Node> path = astar(GRID, startX, startY, targetX, targetY);

        if (path != null) {
            System.out.println("Path found:");
            for (int i = path.size() - 1; i >= 0; i--) {
                Node node = path.get(i);
                System.out.println("(" + node.x + ", " + node.y + ")");
            }
        } else {
            System.out.println("No path found.");
        }
    }
}
```

13. Algorithm Design Strategies

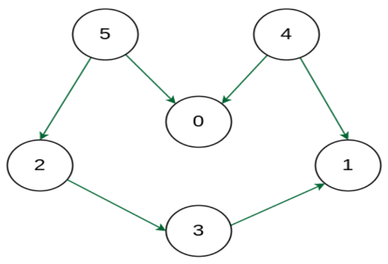
13.1 Topological Sort

Topological sorting for **Directed Acyclic Graph (DAG)** is a linear ordering of vertices such that for every directed edge u-v, vertex **u** comes before **v** in the ordering.

Note: Topological Sorting for a graph is not possible if the graph is not a **DAG**.

Example:

Input: Graph



Output: 5 4 2 3 1 0
Explanation: The first vertex in topological sorting is always a vertex with an in-degree of 0 (a vertex with no incoming edges). A topological sorting of the following graph is "5 4 2 3 1 0". There can be more than one topological sorting for a graph. Another topological sorting of the following graph is "4 5 2 3 1 0" **Approach:**

- Create a **stack** to store the nodes.
- Initialize **visited** array of size **N** to keep the record of visited nodes.
- Run a loop from **0** till **N :**
- if the node is not marked **True** in **visited** array then call the recursive function for topological sort and perform the following steps:
    - Mark the current node as **True** in the **visited** array.
    - Run a loop on all the nodes which has a directed edge to the current node
    - if the node is not marked **True** in the **visited** array:
    - Recursively call the topological sort function on the node
    - Push the current node in the stack.
- Print all the elements in the stack.

```
public class TopologicalSort {

    // Function to perform DFS and topological sorting
    static void
    topologicalSortUtil(int v, List<List<Integer> > adj,
                        boolean[] visited,
                        Stack<Integer> stack)
    {
        // Mark the current node as visited
        visited[v] = true;

        // Write code here
        ………
    }

    // Function to perform Topological Sort
    static void topologicalSort(List<List<Integer> > adj,
                                int V)
    {
        // Write code here
        .........

        // Print contents of stack
        System.out.print(
            "Topological sorting of the graph: ");
        while (!stack.empty()) {
            System.out.print(stack.pop() + " ");
        }
    }

    // Driver code
    public static void main(String[] args)
    {
        // Number of nodes
        int V = 4;

        // Edges
        List<List<Integer> > edges = new ArrayList<>();
        edges.add(Arrays.asList(0, 1));
        edges.add(Arrays.asList(1, 2));
        edges.add(Arrays.asList(3, 1));
        edges.add(Arrays.asList(3, 2));

        // Graph represented as an adjacency list
        List<List<Integer> > adj = new ArrayList<>(V);
        for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<>());
        }

        for (List<Integer> i : edges) {
            adj.get(i.get(0)).add(i.get(1));
        }

        topologicalSort(adj, V);
    }
}
```

13.2 Egg Dropping Puzzle

Given n cities and distances between every pair of cities, select k cities to place warehouses (or ATMs or The following is a description of the instance of this famous puzzle involving N = 2 eggs and a building with K = 36 floors.
Suppose that we wish to know which stories in a 36-story building are safe to drop eggs from, and which will cause the eggs to break on landing. We make a few assumptions:

- An egg that survives a fall can be used again.
- A broken egg must be discarded.
- The effect of a fall is the same for all eggs.
- If an egg breaks when dropped, then it would break if dropped from a higher floor.
- If an egg survives a fall then it would survive a shorter fall.
- It is not ruled out that the first-floor windows break eggs, nor is it ruled out that the 36th-floor does not cause an egg to break.

If only one egg is available and we wish to be sure of obtaining the right result, the experiment can be carried out in only one way. Drop the egg from the first-floor window; if it survives, drop it from the second-floor window. Continue upward until it breaks. In the worst case, this method may require 36 droppings. Suppose 2 eggs are available. What is the least number of egg droppings that are guaranteed to work in all cases?

The problem is not actually to find the critical floor, but merely to decide floors from which eggs should be dropped so that the total number of trials is minimized.

The solution is to try dropping an egg from every floor(from 1 to K) and recursively calculate the minimum number of droppings needed in the worst case. The floor which gives the minimum value in the worst case is going to be part of the solution.

In the following solutions, we return the minimum number of trials in the worst case; these solutions can be easily modified to print the floor numbers of every trial also.

What is worst case scenario?

Worst case scenario gives the user the surety of the threshold floor. For example- If we have ‘1’ egg and ‘K’ floors, we will start dropping the egg from the first floor till the egg breaks suppose on the ‘Kth’ floor so the number of tries to give us surety is ‘K’.

To solve the problem follow the below idea:

Since the same subproblems are called again, this problem has the Overlapping Subproblems property. So Egg Dropping Puzzle has both properties of a dynamic programming problem. Like other typical DP problem, re-computations of the same subproblems can be avoided by constructing a temporary array eggFloor[][] in a bottom-up manner.

In this approach, we work on the same idea as described above neglecting the case of calculating the answers to sub-problems again and again. The approach will be to make a table that will store the results of sub-problems so that to solve a sub-problem, would only require a look-up from the table which will take constant time, which earlier took exponential time.

Formally for filling DP[i][j] state where ‘i’ is the number of eggs and ‘j’ is the number of floors:

We have to traverse for each floor ‘x’ from ‘1’ to ‘j’ and find a minimum of:

(1 + max(DP[i-1][j-1], DP[i][j-x])).

Below is the illustration of the above approach:

i => Number of eggs

j => Number of floors

Look up find maximum

Lets fill the table for the following case:

Floors = ‘4’

Eggs = ‘2’

1 2 3 4

1 2 3 4 => 1

1 2 2 3 => 2

For ‘egg-1’ each case is the base case so the

number of attempts is equal to floor number.

For ‘egg-2’ it will take ‘1’ attempt for 1st

floor which is base case.

For floor-2 =>

Taking 1st floor 1 + max(0, DP[1][1])

Taking 2nd floor 1 + max(DP[1][1], 0)
DP[2][2] = min(1 + max(0, DP[1][1]), 1 + max(DP[1][1], 0))
For floor-3 =>
Taking 1st floor 1 + max(0, DP[2][2])
Taking 2nd floor 1 + max(DP[1][1], DP[2][1])
Taking 3rd floor 1 + max(0, DP[2][2])
DP[2][3]= min('all three floors') = 2
For floor-4 =>
Taking 1st floor 1 + max(0, DP[2][3])
Taking 2nd floor 1 + max(DP[1][1], DP[2][2])
Taking 3rd floor 1 + max(DP[1][2], DP[2][1])
Taking 4th floor 1 + max(0, DP[2][3])
DP[2][4]= min('all four floors') = 3

```java
import java.io.*;

class EggDrop {

    // A utility function to get maximum of two integers
    static int max(int a, int b) { return (a > b) ? a : b; }

    /* Function to get minimum number of trials needed in worst
    case with n eggs and k floors */
    static int eggDrop(int n, int k)
    {
           //Write Code Here
      …     ….    ….    ….
    }

    /* Driver code */
    public static void main(String args[])
    {
        int n = 2, k = 36;
        System.out.println(
            "Minimum number of trials in worst"
            + " case with " + n + "  eggs and " + k
            + " floors is " + eggDrop(n, k));
    }
}
```

13.3 Policemen Catch Thieves

Given an array of size n that has the following specifications:

1. Each element in the array contains either a policeman or a thief.
2. Each policeman can catch only one thief.

3. A policeman cannot catch a thief who is more than K units away from the policeman.

We need to find the maximum number of thieves that can be caught.

Algorithm:

1. Initialize the current lowest indices of policeman in pol and thief in thi variable as -1.
2 Find the lowest index of policeman and thief.
3 If lowest index of either policeman or thief remain -1 then return 0.
4 If |pol – thi| <=k then make an allotment and find the next policeman and thief.
5 Else increment the min(pol , thi) to the next policeman or thief found.
6 Repeat the above two steps until we can find the next policeman and thief.
7 Return the number of allotments made.

```
// Java program to find maximum number of thieves caught
import java.io.*;
import java.util.*;
class GFG {
        // Returns maximum number of thieves that can be caught.
        static int policeThief(char arr[], int n, int k)
        {
                // Code Here
        .....   .....   ....            }

        // return 0 if no police OR no thief found
                if (thi == -1 || pol == -1)
                        return 0;
        // loop to increase res if distance between police and thief <= k
                while (pol < n && thi < n) {
                        // Write Code Here
                .....   ......  .....}
        }

        // Driver code starts
        public static void main(String[] args)
        {       char arr1[] = { 'P', 'T', 'T', 'P', 'T' };
                int n = arr1.length;
                int k = 2;
                System.out.println("Maximum thieves caught: " + policeThief(arr1, n, k));

                char arr2[] = { 'T', 'T', 'P', 'P', 'T', 'P' };
                n = arr2.length;
                k = 2;
                System.out.println("Maximum thieves caught: " + policeThief(arr2, n, k));

                char arr3[] = { 'P', 'T', 'P', 'T', 'T', 'P' };
                n = arr3.length;
                k = 3;
                System.out.println("Maximum thieves caught: " + policeThief(arr3, n, k));
        }
}
```

13.4 Fitting Shelves Problem

Given length of wall w and shelves of two lengths m and n, find the number of each type of shelf to be used and the remaining empty space in the optimal solution so that the empty space is minimum. The larger of the two shelves is cheaper so it is preferred. However cost is secondary and first priority is to minimize empty space on wall.

A simple and efficient approach will be to try all possible combinations of shelves that fit within the length of the wall.

To implement this approach along with the constraint that larger shelf costs less than the smaller one, starting from 0, we increase no of larger type shelves till they can be fit. For each case we calculate the empty space and finally store that value which minimizes the empty space. if empty space is same in two cases we prefer the one with more no of larger shelves.

```
// Java program to count all rotation divisible by 4.
public class GFG {
	static void minSpacePreferLarge(int wall, int m, int n)
	{
		// For simplicity, Assuming m is always smaller than n initializing output
variables
		int num_m = 0, num_n = 0, min_empty = wall;
		// p and q are no of shelves of length m and n rem is the empty space
		int p = wall/m, q = 0, rem=wall%m;
		num_m=p;
		num_n=q;
		min_empty=rem;
		while (wall >= n) {
			// Code Here
		...	.....	....
			}
		System.out.println(num_m + " " + num_n + " " + min_empty);
	}

	// Driver Code
	public static void main(String[] args)
	{
		int wall = 24, m = 3, n = 5;
		minSpacePreferLarge(wall, m, n);

		wall = 24;
		m = 4;
		n = 7;
		minSpacePreferLarge(wall, m, n);
	}
}
```

14. Final Notes

The only way to learn Design and Analyzing algorithm is writing algorithms for solving real time applications and challenging problems optimally through programming. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

VI.REFERENCE BOOKS:

1. Levitin A, "Introduction to the Design and Analysis of Algorithms", Pearson Education, 2008.
2. Goodrich, M.T. R Tomassia, "Algorithm Design foundations Analysis and Internet Examples", John Wiley and Sons, 2006.
3. Base Sara, Allen Van Gelder, "Computer Algorithms Introduction to Design and Analysis", Pearson, 3rd edition, 1999.

VII. ELECTRONICS RESOURCES

2. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
3. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
4. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

VIII. MATERIALS ONLINE

3. Course template
4. Lab Manual

V.TEXT BOOKS:

1. Karin R Saoub, *Graph Theory: An Introduction to Proofs, Algorithms, and Applications*, 1st edition, Chapman and Hall, 2021.
2. S S Sastry, *Introductory Methods of Numerical Analysis*, 5th edition, 2012.

VI.REFERENCE BOOKS:

4. Levitin A, "Introduction to the Design and Analysis of Algorithms", Pearson Education, 2008.
5. Goodrich, M.T. R Tomassia, "Algorithm Design foundations Analysis and Internet Examples", John Wiley and Sons, 2006.
6. Base Sara, Allen Van Gelder, "Computer Algorithms Introduction to Design and Analysis", Pearson, 3rd edition, 1999.

VII. ELECTRONIC RESOURCES

1. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
2. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
3. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

WEB SYSTEMS ENGINEERING LABORATORY								
IV Semester: CSE / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
		L	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
ACSD17	Core	0	0	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisites: Programming with Objects								

I. COURSE OVERVIEW:
This course will give you the basic terminology and fundamental concepts to build modern web applications. This course introduces students to develop web applications. This course presents the basics of HTML5 and CSS3 for Web application development using HTML links and HTML forms. Introduction to the use of React router and its use in developing single-page applications, redux to develop React Redux powered applications, client-server communication and the use of REST API on the server side and react primitives render to native platform UI. This course will make the students to expose the front-end framework Bootstrap and to basic security mechanisms for server-side web application development.

II. COURSES OBJECTIVES:
The students will try to learn

I. The characteristics, systematic methods, model for developing web applications.
II. The concepts of client side programming with Bootstrap, JavaScript, Ajax, Design user interfaces that follow best practices for usability and user experience
III. Web application Development Database with React and React Native.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO1	Create a web page with different layouts including links by applying different styles and colors to produce specified outputs
CO2	Develop a responsive web application using bootstrap with background images, menus, with admin panel and tables.
CO3	Develop interactive forms with different styles using javascript, CSS and bind data using AJAX
CO4	Develop single page applications using react router and make use of react data libraries for data visualization in dynamic pages
CO5	Adapt to design and develop web applications like drunken snake game and chat application with API responses using the industry's current models and architectures
CO6	Test for the database to extend the features and deployment of applications for solving problems that require interaction with a web server

IV.COURSE CONTENT:

Getting Started Exercises

1 CEW (Corporate Employee Welfare) Net Work

Objective:

All corporate employees can share following value points using CEW web base application

1. Innovative thoughts
2. All Company Events (photos share)
3. Technical droughts
4. Work experience
5. Personal relating things (Property Sales and Home Rent)
6. All company growth (share market)
7. All company Employee referral (job vacancy)
8. Matrimonial
9. Birth days
10. Travel / Picnic / Get together Plans

Functional Requirements

1. Employee verification
2. All Company Events
    - Technical droughts
    - Work experience
    - Property sales and Home rent
    - All company growth(share market)
    - All company Employee referral(job vacancy)
    - admin
    - User management

Reports

1. Admin report
2. Bank report
3. Viewable and downloadable reports with password protection

See also for authentic understanding, click the link

https://cewacor.nic.in/MasterStatic/Employee_Welfare_Cell

https://www.nexteraenergy.com/employee-central.html

https://ongcindia.com/web/eng/career/why-work-with-ongc/employee-welfare

https://www.pelagohealth.com/resources/hr-glossary/employee-welfare/#

Each student has to refer any one of the web sites stated above.

2. E – Healthcare Advisor

Objective: The main objective of this project is to implement a computer-based Healthcare Information System. This system will help the users to identify certain diseases by answering certain questions asked by the system. Based on the diagnose received the user will be getting some suggestion of medicines that are available at the local chemist without prescription with an advice to visit the doctor. The system once ready should be able to train itself with the feedback given to it (Artificial Intelligence). The database will be developed with open source software.

- To conduct a diagnose in order to identify the disease
- To design a healthcare management system
- To maintain patient history and system keep self learning(artificial intelligence) to update the database.

Functional Requirements

1. It focused on the acquisition and management of disease database
2. It mainly emphasized creation and implementation of patient and disease management information system.
3. It will ease and speed up the planning decision making process , secure confidential and reliable reports
4. It helps for addressing problems of security secrecy and confidentially of patients.
5. Used to check the delays errors inconsistencies in medical records and access to historical records
6. To maintain patient profile

Reports

1. Search the name, place, disease, periodic base reports
2. Search the nearest government approved Clinic/Hospital and suggest it to the patient.

See also for authentic understanding, click the link

https://codethislab.com/website-health-advisor/
https://www.ehealthsystem.com/
https://e-clinic.co.uk/

Each student has to refer any one of the web sites stated above.

3 HCSS (Highly confidential Security System)

Objective: Due to busy life style we can't remember all confidential data like All maid Id, Password, All bank account no, Insurance policy No, PAN NO, Driving License No, Password Port No, All education certificate Numbers, Some highly value scan copy, some confidential photo and music ,videos. It is a software locker which can hold user sign in information, much like the functionality of SSO, however one can also couple a biometric hardware to strengthen the security system. The system will help user in logging in to the client system for which it is holding/storing the password, either by the software interface or directly by hardware interface. So we can develop highly security web application (new security algorithm and hardware system) .so we can store all confidential data in single credentials

Functional Requirements

1. Security System (Software and hardware)
2. To provide mail id and password locker
3. Bank account information locker
4. Videos locker v. Images locker
5. Music locker
6. Admin
7. User management

Reports

1. Admin report
2. Bank report
3. Viewable and downlable reports with password protection

See also for authentic understanding, click the link

https://azure.microsoft.com/en-in/solutions/confidential-compute

Each student has to refer any one of the web sites stated above.

4 Food Safety Portal

Objective: Provide an application which allows residents and visitors of a city to find out more about food-related aspects about the city. Aspects include information about restaurant and market certifications and what to look for (education), local diet and/or delicacies, or even a restaurant guide.

Functional Requirements

1. Web accessible information base
2. Provide templates for information entry – e.g. education, dining guide, food handling guide, etc.
3. Allow for easy update of information by city employees
4. Allow for easy retrieval of feedback collected to facilitate acting on feedback received
5. Extensible to allow each city to update with their own specific information
6. Allow report of food-related illness (non-life-threatening)
7. Allow report of food safety concern
8. Allow submission of suggestions for improvement
9. Enable a map view of the city which marks locations of restaurants and markets

Reports

1. Report of restaurants visited and reviewed
2. Report of food-related illnesses
3. Usage report of website (hit rates, popular pages)
4. Report of user suggestions

See also for authentic understanding, click the link

https://services.india.gov.in/service/detail/national-food-security-portal-1
https://www.india.gov.in/content/national-food-security-portal
https://www.ifpri.org/publication/india-food-security-portal
https://damoh.nic.in/en/rationmitra/

Each student has to refer any one of the web sites stated above.

5. HR Operations Manager

Objective: In business environments filled with diverse forms of content, continually changing and complex business processes, and an array of different lineof-business interfaces, companies are looking for ways to gain operational efficiencies, reduce risk, and improve quality through exception management and end-to-end process visibility.
The solution will be used to demonstrate how value can be delivered across any business process by linking business processes to relevant business content, and to the people that need to make business decisions.

Functional Requirements

1. Workflow capabilities of the proposed system

2. Components for Business Process
3. HR new hiring process with;
   i. Applicant submits his/her Resume to an identified e-mail id with an opportunity id.
   ii. On receiving the email, extract the mail contents & store it.
   iii. Based on the opportunity id send it to respective HR. (Finance, IM, etc.)
   iv. HR will verify the resume; if shortlisted, forward it to respective Managers. On rejection, an email will be sent to the Applicant with reason.
   v. Manager will schedule the interview & assign a team member to perform interview.
   vi. Interviewer conducts the interview & updates the interview results.
   vii. Manager will update based on the results.
   viii. HR will send out the offer letter if selected.
   ix. Applicant will respond with Acceptance/Rejection subject line. On Rejection process will be ended. On Acceptance new employee ID/mail-id will be generated & process will be ended.

Reports

1. Reports customizing the stored data in a platform independent format and displaying it using style sheets.
2. Admin must be able to data in reports in excel sheets.
3. Admin must be able to make pictorial depiction of data in excel sheets for better understanding.
4. Reports should be elaborate for all the users.

See also for authentic understanding, click the link
https://azure.microsoft.com/en-in/solutions/confidential-compute

Each student has to refer any one of the web sites stated above.

6. Investor self service Application

Objective: To build an investor self service portal for an Asset Management Company (AMC) that would allow its investors to have real time access to customer portfolio and various other functionalities.

Functional Requirements

The investor self services portal (ISSP) will provide the following functionality to the AMC's investor.

1. Login – Login in to the AMC's investor portal with folio number and pin.
2. Account Summary – Portfolio details with cost value and current valuation.
3. Account Statement – Ability to generate account statement in predefined format.
4. Fresh Purchase - Ability to sign up as a new investor by entering relevant data and having the application form prefilled, ready to be signed and dispatched to the AMC. Ability to track status of Application using Application Inquiry Screens. Ability to make first purchase online (through PG integration) or through cheque/demand draft.
5. Additional Purchase – Ability to invest into new schemes or make additional investments in an existing scheme. This module will be integrated with the payment gateway. Customer account will be debited and PG provider's pool account will be credited instantly. Purchase transaction will be booked only after successful payment confirmation from the PG provider.

6. Redemption – Ability to redeem existing investments. Redemption can be done for all units, selected no of units or by specifying amount.
7. Switches – Ability to switch from one scheme to another. It will be possible to switch to new schemes as well.
8. Change of Dividend option – Ability to change Dividend option from Re-Invest to Payout and vice versa. Not applicable for Growth schemes/plans.
9. View Transactions – Listing of last few transactions initiated by the investor.
10. View Bank Details – Details of existing bank mandates specified by investor while booking purchases.
11. My Profile – Listing of the investors profile including his address, email id, registered PAN No, KYC status for himself and joint holders etc.
12. Change pin – Investor can change his pin after having logged in to the portal.
13. Security features should include Account lockout – Account can be locked out after a configurable number of failed login attempts. There must be a capability to unlock the folio from the application console.
14. Security features should include SSL based access – The transport channel will be SSL enabled right from the client browser. Access to the PG provider's site will also be SSL enabled.
15. Security features should include Change Pin on first access – Pin will have to be changed by investor on first login.
16. Security features should include Regular 'Change Pin' Mechanism - Subsequently, investor can be forced to change his pin at a configured time interval (say 3 months)
17. Account debit to be authorized by investor by entering his internet banking credentials along with any additional security imposed by the bank.
18. ISSP will perform a number of basic validations on the transactions. Examples of local validations would be:
    i. Minimum investment amount
    ii. KYC status and PAN status
    iii. Inability to invest in a closed ended scheme where the scheme has already been closed.
19. Administration Module will include the functionalities:
    i. Changes in the Masters will be made by the Operations Team
    ii. Locking and Unlocking of Investor Accounts
    iii. Viewing all transactions across all investors

Reports

1. Reports customizing the stored data in a platform independent format and displaying it using style sheets.
2. Admin must be able to data in reports in excel sheets.
3. Admin must be able to make pictorial depiction of data in excel sheets for better understanding.
4. Reports should be elaborate for all the users.

See also for authentic understanding, click the link

https://www.tricorglobal.com/services/investor/self-service-portal/

https://www.infosys.com/industries/utilities/case-studies/web-self-service.html

https://www.salesforce.com/products/customer-self-service/

Each student has to refer any one of the web sites stated above

7. E-MANDI (electronic -vegetable market)

Objective: The main objective of this project is build a website which will help civilian, retailer whole seller and even the farmer to get the best from his inputs. with the help of this a farmer will be able to know the best value for his vegetable and will not be fooled by the marketers. it will help in keeping the transparency between the whole seller and retailer and also the selection for civilian for his requirement become easy .so this will help in eradicating black marketing and inflation.

Functional Requirements

1. People can register to have a complete view of the market including the pricing of vegetables, pricing difference between whole seller and retailer, actual pricing stated by government, best possible retailer in the market for civilian in his area, revenue generated last month and a period of time.
2. Non registered can have an overview of these facilities excluding some.
3. Feedback or complaint facilities directly connected to government bodies to keep a view on the market but mentioning the unique id of the complainer.
4. Admin should be able to see all record from any users.
5. The records shown for selling should be available in a format of Quantity name, Quantity available, price
6. The database should be robust enough to handle all the online transactions which will be happening.
7. Website will be available in regional languages.
8. People will have facility of viewing the price difference between different regions and the inflation rate also.
9. Person have facility of booking vegetables for commercial use on a large scale online through bigger marketers.
10. Help section for those who are unable to understand the website or any of its part.

Reports

1. Daily report of enrollment to Admin.
2. Monthly report of enrollment as per states to Ad-min i
3. Work hours uses of Computer professionals on a monthly basis to Admin

See also for authentic understanding, click the link

https://www.jiomart.com/c/groceries/fruits-vegetables/fresh-vegetables/229?prod_mart_groceries_products_popularity%5Bpage%5D=2
https://www.bigbasket.com/cl/fruits-vegetables/
https://farmersfz.com/

Each student has to refer any one of the web sites stated above.

8. Issue Tracking scenario

Objective: To provide Issue tracking for projects

Functional Requirements

1. Individual accounts for Developers.

2. Ticket creation and updation.
3. Search for tickets.
4. Set Priorities for Tickets (higher-1,2,3)
5. View ticket details.
6. Assigning or UN-assigning a ticket to Developer by higher authorities or by themselves.
7. Uploading patch files or any other required files after solving the issue and update the ticket status.
8. Maintain activities for
   I. Ticket- Comments and History.
   II. Developer – Comments and work log.
9. Export a ticket in different formats like doc and pdf.

Reports

1. Daily Tickets Reports
2. Daily Solved tickets Reports
3. Monthly Tickets Reports

See also for authentic understanding, click the link

https://www.projectmanagement.com/wikis/233063/issue-tracking#_=_
https://www.zoho.com/bugtracker/issue-tracking-system.html
Each student has to refer any one of the web sites stated above

9. Online grievance redressal system

Objective: The people need not go to the higher authorities always when they face problems. They can use the service of this software browser and can give their complaint and the complaint is taken up by the employee of specified department and he solves the problem.

1. To create a user-friendly online interface for citizens to communicate with administrative body and, reduce the distance and time barrier between citizens and administration.
2. To create a online platform where people can share ideas, invoke discussions, issue complaints, create suggestion/petitions for improvement of city administration.
3. To encourage the citizens to actively participate in city administration to bring transparency and flexibility in system.

Motivations to build this system:

1. Limited hour service availability in the current system.
2. Lack of involvement of people in exhibiting their responsibilities towards society.

Functional Requirements

1. Online Registration for public
2. User-friendly Interface
3. Easy intake of user need
4. Manager, employee related user id, passwords are send to their respective mails
5. Verification of manager, employee, public details
6. Online interaction of administrator, employee and managers

7. End to end interaction of employees with public
8. Administrator controls all department queries

A. Users (Citizens):

- I. Users should be able to create new account, log-in to their existing accounts which will give them
- II. The authority to use the services provided by the system.
- III. Authenticated users should be able to issue complaints, check complaint status, submit feedback, browse through other complaints and their feedback.
- IV. Authenticated users should be able to create suggestions/petitions; other users can support or make suggestions for petitions; forward petitions to corresponding authority for possible implementation.
- V. Users can to create groups where users can share their experiences; discuss common problems, and the possible solution;

B. Municipal authorities:

- I. Municipal authorities can log-in to their accounts as created by administrator.
- II. Authorities can access all the complaints, suggestions from users.
- III. Invoke proper activity in response to valid complaints, or redirect inappropriate complaints to the administrator.
- IV. Give response to complaints with activity reports.

C. Administrators:

- I. Create, and monitor accounts of authorities.
- II. Filter the content reported as inappropriate and handle threats.
- III. Handle complaints about improper response by municipal authorities.

D. NGO's:

- I. NGO can form user groups similar to other users.
- II. NGO's can publicize their social causes on the site.

Reports

1. Weekly report
2. Monthly report
3. Status report
4. Yearly report
5. Escalation reports based on responsibility matrix
6. Queries and responses answered report
7. Complaint report including complaint details, response details, feedback
8. Section-wise user-feedback summary

See also for authentic understanding, click the link
https://pgportal.gov.in/
https http://grs.nios.ac.in/
https://www.consumercomplaints.info/?tm=tt&ap=gads&aaid=adaVTn2qEIBda&gclid=Cj0KCQjw-pyqBhDmARIsAKd9XIOOudygMWMedVUYZQpHlS1BUBOaBTvrdlTyXMaOqBqsIc2q47YKGHIaAs3zEALw_wc

Each student has to refer any one of the web sites stated above.

10. E-Hospital System

Objective: A solution that allows all the E-Hospital System

Functional Requirements

1. Patients should be able to check doctor availability and book the appointment.
2. Doctor should be able to accept or reject appointment.
3. Doctor prescription storage and viewable to all departments.
4. Lab admins to update lab test results
5. Pharmacists to provide stock details and delivery of medicines.
6. Billing user to bill based on all the transactions done at each department and keep a track of the same.
7. Automatic notification between doctor, patient, pharmacists, lab admins and billing divisions based on use case.

Reports

1. Doctor availability reports
2. Day wise Patients reports
3. Pharmacists delivery report

See also for authentic understanding, click the link

https://www.adroitinfosystems.com/products/ehospital-systems/
https://nextgen.ehospital.nic.in/ehosModuleList

Each student has to refer any one of the web sites stated above

11. Smart Transportation Based Car Pooling System

Objective: To maintain a web based intranet application that enables the corporate employees within an organization to avail the facility of car pooling effectively.

Functional Requirements

1. A system for an Admin who can enter the employee details like name, contact number, vehicle details etc.
2. Corporate employees can register the details to the website
3. The facility to see the available services in the route
4. Employees receive SMS alerts regarding the route and timings.
5. The facility to check whether the vehicle and driver is authorized or not
6. Admin can view the report of the car pooling process to improve the system
7. Employees can report suggestions/complaints in the website
8. Admin can monitor every activity which is performed by system
9. Employees can view the details of registered vehicles and the owners which will develop trust and understanding among the employees

Reports

System will generate

1. Monthly Reports
2. Weekly Reports

See also for authentic understanding, click the link

https://quickride.in/

https://www.pcb.ub.edu/en/carpooling/

Each student has to refer any one of the web sites stated above

12. Law & Order Automation

Objective: To deliver next generation police and law enforcement reporting tools, and setting up intelligence platforms that agencies use to take incoming incident reports, lessen employee resources and allow these enforcement agencies to reallocate resources to much needed community areas

Functional Requirements

1. Administrator should be able to create/edit a virtual police station (PS) which represents a real police station as a first time setup.
2. Appointing of police officers to a particular police station which is present in a specific zone or to a specific district as a first time setup, he should be transferable at later time.
3. PS should have areas of control which can be modified at later time.
4. Police station has several departments like Law and Order, Women Protection, Cybercrime, Traffic and control, CBI, etc. Separate module for each dept would be needed.
5. When a complaint is made it undergoes various processes like FIR, Charge Sheet, Property Seizure, court disposal etc all these activities are performed by a PS.
6. Traffic and Control has important part where the Traffic inspector would be filing a charge sheet from a mobile or PDA.
7. Maintaining the criminal information state wise/area wise/age wise is mandatory
8. Sharing of case details with PS in other states is needed. Note : Other state may use different database and different platform (use of IBM MQ is necessary)
9. Communication between officers is mandatory through forum, chat, polls.
10. The magistrate should be able to access the case details and provide/deny the arrest warrant.
11. Citizens should be able to apply for various licenses like Arms, loud speaker, Hotel/lodge, browsing centre, mass meetings etc., and the officer should be able to approve/reject which will be notified to the applicant via SMS and Mail
12. Secured registration of citizens is needed where they need to provide proof of citizenship, which will be cross checked by the police officer of that area.

Reports

1. Complaints filed in a day and action taken to it. It should also report unattended complaints.
2. Crime rate due to various types of crimes in a month/year and also in district/state wise.
3. Report regarding most wanted criminals and bounty information if available.
4. Police officers often export the FIR copy to PDF format.

See also for authentic understanding, click the link

https://www.mha.gov.in/en/commoncontent/law-and-order
https://nclt.gov.in/
https://lawcommissionofindia.nic.in/
https://doj.gov.in/

Each student has to refer any one of the web sites stated above

13. Online Book Sales with Mobile SMS

Objective: The main objective of this project is to implement a computer based Online Book Sales System with the help of Mobile SMS. This system will help the users to know automatically when certain book requested by them has arrived once they have registered with the website. It will also help the users to know the latest books of their interests being introduced into the market like fiction, science, technology, romance etc. The database will be developed with open source software.

- To develop a book management system
- To design a Book Sales Management system
- To design a customer Mobile-Book sales management system interface which interacts with database system
- To mobile client be optimized and also an interface for connecting thru PC

Functional Requirements

1. It focused on the acquisition, distribution and management of books.
2. It mainly emphasized creation and implementation of a book sales management information system.
3. It automated the system with the help of SMS that informs the user about the availability, price and method of purchasing the book.
4. It will ease and speed up the planning decision making process process, secure confidential and reliable reports
5. It help for addressing problems of security secrecy and confidentially of customer records.
6. Used to check the delays errors inconsistencies in records and access to historical records.
7. It has eased the control and distribution of books in various parts of the country basing on regional demands

Reports

To be thought of according to requirement

See also for authentic understanding, click the link

https://www.amazon.in/Books/b?ie=UTF8&node=976389031
https://www.smsindiahub.in/bulk-sms-for-book-stores/

Each student has to refer any one of the web sites stated above

14. E-Resource Technology

Objective: "E-Resource Technology" is education based website/software, helping students to get all resources & study materials of every courses available. It uses "E-Book" facility. It is reliable & time efficient approach compared to all links of the website provided by any search engine while searching for course materials.

Functional Requirements

1. To provide official & legal links of the website from which user (student) can download resources & study materials of relevant course
2. Only accessible after registering to that specific website
3. Getting associated with professors of esteemed institutes & colleges
4. To provide interaction between professor & students (users)
5. Getting associated with well known publication house so as to students can access soft copy of books published for future use
6. Not all links provided by GOOGLE are relevant. This would provide a better service.
7. Not all links provided by GOOGLE are virus free or recommended to download. This would be a better approach.
8. To implement this time efficient strategic

Reports

1. Students/Users Report
2. Professors of Esteemed Institutes & Colleges report
3. Toppers of College / Universities report
4. Mentors report

See also for authentic understanding, click the link

https://library.iitd.ac.in/ERD
https://www.education.gov.in/e-contents
https://www.washington.edu/doit/making-electronic-resources-accessible-libraries

Each student has to refer any one of the web sites stated above

15. Employee Expense management System

Objective: To build an automated employee expense management

Functional Requirements

1. Voucher Entry – Screen for entering expense vouchers for any reimbursable expenses borne by the employee. A voucher should have one header and multiple lines providing detailed information of expenses incurred along with amounts.
2. Multi-level workflow – Based on the Employees department and designation, the vouchers should flow to his supervisors for their approvals. The number of approvals required will vary according to total amount of the voucher and approval limits set for supervisors.
3. Case Management – Every voucher should start a new process instance (case). Managers should be able to view vouchers waiting for their approval in their Inbox by logging in to the application. Employee should be able to track progress of his vouchers.
4. Draft Vouchers – System should provide the ability to save incomplete vouchers and submit them for approval later after completion.
5. Rejection flow – At any point in the workflow, managers should have the option of rejecting any voucher stating appropriate reasons. Rejected vouchers should come back to the Employee, who is then allowed to change details on the vouchers or provide additional information required for the approval.
6. Accounts View – Accounts department users should be able to view approved vouchers of all employees and mark vouchers as paid. This step completes the lifecycle of the voucher and the associated process instance.

Reports

1. MIS Reports – MIS reports stating status of the vouchers
2. Exception Reports – Reports listing vouchers awaiting approval above a defined turn-around time (threshold)
3. Master data management – Creation of masters for Employee Hierarchy, setting approval limits etc

4. Employee Hierarchy View - View of Employee Hierarchy
5. Rejected vouchers report – Report listing vouchers rejected by Managers
6. Reports customizing the stored data in a platform independent format and displaying it using style sheets.
7. Admin must be able to data in reports in excel sheets
8. Admin must be able to make pictorial depiction of data in excel sheets for better understanding
9. Reports should be elaborate for all the users.

See also for authentic understanding, click the link

https://use.expensify.com/expense-management

Each student has to refer any one of the web sites stated above

16. Insurance System with Tracking Manager

Objective: This project provides five types of Insurance services, which includes Life Insurance, medical Insurance, Motor Insurance. Home Insurance, Travel Insurance. This project provides loan facility for Motor Purchasing. The details can be viewed and updated by the officials of the company.

Functional Requirements

A. User Panel

I. A User can view the details of various policies and schemes offered by the Insurance Company.
II. New Users can register with the site so that he can get information online.
III. An existing policyholder can view his policy details and calculate the premium.
IV. The web site provides information about the new strategies and subsidiary schemes of the company.
V. Provides loan facility for policyholders and online payments.
VI. Provides Loan EMI calculator
VII. Provides Interest calculator

B. Administrator Panel

I. Administrator gives the approval for the new users
II. Administrator edit modify and delete, upload certain information
III. Provide the facility to send the statements and loan EMI details in pdf format to users mail ids.

Reports

1. Interest Calculation
2. Month wise registered users Insurance
3. Region wise registered users Insurance
4. Month and Region wise list of users for loan
5. Monthly Premier
6. Monthly Loan EMI

See also for authentic understanding, click the link

https://uiic.co.in/

https://nationalinsurance.nic.co.in/

Each student has to refer any one of the web sites stated above

17. AIR TRANSIT TRIP PLANNER

Objective: Build the air transit trip planner which This passenger on transit will spend their time at the airport while waiting for the next flight without any planning to visit tourism places nearby due to short period of time and much information needed to plan a short trip. This is such a waste because they actually can boost countries' economy in the tourism sector if they spend their time to visit some tourism places nearby the airport. Because of that, Air Transit Trip Planner aims to help these passengers on transit by planning a short trip to the nearby tourism places within the transition hour they have.

Air Transit Trip Planner also using Google Map API as a map to guide user and Google Places API as a data center to grab all the tourism places and its details. The output of this calculation is a suggestion of short trip planner for the user (passenger on transit). The trip planner will list down the tourism places user can visit within their flight transition hour, the distance to go there and also time taken to go there. Indirectly, this Air Transit Trip Planner application also helps to boost economy in the tourism sector of a country

Functional Requirements

1. A 'Main Page' which the user will see when they launch the application. In the background when this interface prompts, the system actually will automatically detect user current location by using GPS or Google Places. If no internet access or GPS is turn off, the system will prompt a pop up asking the user to turn on GPS or connect to internet. But if everything is fine, user just need to key in their transit hour and departure time then click enter.
6. A 'Tourism places' page will calculate how many tourisms places the user can visit within the transit hour they have by using the algorithm.
7. A 'place details" page will direct to new user interface with scrollable list view.
8. A 'pop up' page when user click each mark on map. When the user clicks the mark, the pop up will show the address of each location.
9. A 'GPS Navigation' page is using satellite and it is implementing context awareness concept where it will direct user to their destination. This interface also has voice navigation which will help the user to hear the navigation while driving rather than looking at the map continuously.

Reports

1. Passenger Report
2. Flight timing Report
3. Near by tourism places Month Report

See also for authentic understanding, click the link

https://travelplanner.co.in;
https://www.expedia.com;
https://www.orbitz.com;
https://www.kayak.com and so on.

Each student has to refer any one of the web sites stated above.

18. Restaurant Reservation and Table Management Solutions

Objective: Restaurant Reservation and Table Management system is a Web application designed to help you (and your coworkers) to select the restaurant you are going to eat in. You can manage users, restaurants, menus, prices, give notations to each lunch, etc.

Functional Requirements

1. Reservation Management

a. Easily enter or modify reservations while viewing guest histories.

b. Capture phone numbers, email and mailing addresses.
c. Allow management blocking and VIP pre-assignments.
d. Reduce no-shows with enhanced customer tracking.
e. Take reservations from your website or Open Table 24 hours.

2. Table Management

a. Maximize seat utilization with walk-in and waitlist functionality.
b. Instantly track covers for more efficient kitchen and server management. Increase table turns by tracking party status. Store multiple reservation sheets for holidays and special events.
c. Hold and combine tables for large parties.
d. Record and view shift notes for each day

3. Guest Management

a. Identify regulars and VIPs
b. Track customer preferences to meet and anticipate special requests
c. View customer reservation histories at-a-glance
d. Track special occasions such as guest birthdays and anniversaries Marketing Management
e. Conduct powerful email marketing campaigns to increase repeat business.
f. Print mailing labels to reach select target audiences.
g. Track and reward concierge business.

4. Increase control

a. Manage reservations from the back-office or any other location. Simultaneously control multiple restaurants from key centralized locations.
b. Share guest data across sister restaurants.

Reports

1. Day wise customer Report
2. No. of tables filled Report
3. VIP's Report

See also for authentic understanding, click the link

https://www.elluminatiinc.com/restaurant-reservation-system/;
https://restaurant.opentable.com/;
https://www.tornosubitodubai.com/;
https://indiarestaurant.co.in/ and so on.

Each student has to refer any one of the web sites stated above

19. Stay safe women security application

Objective: Build stay safe women security project is used to provide highly reliable security system for the safety of women. The proposed system is based upon advanced sensors and GPS. The basic aim of the system is to develop a low cost solution for GPS based women tracking system (Women Safety System). The main objective of the system is to track the current location of the person which has an android enabled mobile by extracting the longitude and latitude of that target person.

Functional Requirements

1. A 'Scream Alarm' page used perfect for the females as well as other users that need some kind of safety alarm in case they found out that someone is following or stalking them. It also consists of two

other types of scream alarm. It's an initial distraction which will buy some time and allow the user to escape from the trouble.

c. Male voice scream
d. Police siren.

The user could select one of his/her choice from the "Settings" of the application, as keeping in mind the two other scream alarms are also added in this application as now a days safety and security is everybody's concern.

2. A 'Fake Call Timer' page which the fake call timer allows the user to make fake calls in the time of need. It helps user to escape from an undesirable situation citing an important call from anyone who needs him/her urgently and rest depends upon user creativity. This feature also helps the user to escape from boring social events

   In order to make a fake call the user have to select the "Fake Call" icon and after that user could write any name from which he/she wants a fake call. User could also set up the timer as per the requirement. The user could also set the default timer from the "Settings" icon of the application.

   In a critical situation, the user just have to long term press the fake call button and automatically get a fake call as per the desired selected timer in the settings.

3. A 'Where Are You' which is used to find track friend. While first request is send by the sender. The sender will have to select the "Where Are You" icon and then a new dialog box of "Pick a Friend" will open up. The sender could select any friend and the request will be sent to the receiver. The receiver will accept that request from their end and a message will be sent to the receiver with the present location of the user.

4. A 'Track Me' which will track the user to view the exact dynamic location of the victim. First user have to send the Track Me request at the receivers end. The receiver will accept the request and then his/her name will appear on the friends you are tracking on the bottom of the application. The user could select that friend from there and then it will get automatically re-directed to the Google maps from where the user could view the exact location of the victim and also where's he/she heading to.

5. A 'Friends List' page which shows all the contact numbers of family and friends which are added by the user through contacts. This could be done by selecting the contact icon on the bottom right corner of the friends list.

6. A 'Settings' page which consists of the following features -:

   e. A 'Emergency Services' page allows the Stay Safe Application to send emergency notifications and SMS with the exact location to the emergency contacts.
   f. A 'Low Battery Alert' page alert feature allows the Stay Safe Application to send low battery alert and SMS to the emergency contacts.
   g. A 'Set Scream Sound' page which the user could select any scream sound as per the requirement.
   h. A 'Fake Call Timer (On Long press)' page which the user could set the fake call default timer as per the requirement.

7. A 'Emergency Distress Signal (SOS)' which the distress signal will be generated by the user in case of an emergency. In order to generate the distress signal the user have to shake up his/her phone, then a distress signal will appear at the user end with a default timer of 5 sec. In the end distress signal will be sent to the emergency contacts added by the user at the time of registration. The application send SMS and user details as well as the exact location of the user through a push notification at the receiver end, before sending a distress signal the user first have to turn on the emergency services from the settings of the application.

Reports

1. Day wise women registered Report
2. emergency notifications Report

See also for authentic understanding, click the link

https://womensafetywing.telangana.gov.in;
https://nirbhayaapp.com; https://safetipin.com;
https://wsww.smart24x7.com and so on.

Each student has to refer any one of the web sites stated above

20. Secure Stock Exchange System using Web Services

Objective: This project implements a stock exchange is simply a system that is designed for the sale and purchase of securities of corporations and municipalities. A stock exchange sells and buys stocks, shares, and other such securities. In addition, the stock exchange sometimes buys and sells certificates representing commodities of trade

Functional Requirements

1. Stock Markets & Investments

a. Stock Exchange Listing
b. Stock Options & Analysis
c. Stock Market Crash
d. Selling Stock Certificates
e. Stock Market Forecasts

2. Stock Options

Types of Stocks
a. Stock Option Valuation
b. Restricted Stock Options

3.Related Information

a. Day Trading Stocks
b. Stock Quotes & Stock Ticker
c. Stock Charts
d. Share Portfolio Management

Reports

1. Day wise sellers Report
2. Day wise buys Report
3. Shares report

See also for authentic understanding, click the link

https://stocksandsecurities.adityabirlacapital.com/;
https://www.nseindia.com/;
https://www.jpx.co.jp/english/;
https://www.mstock.com/ and so on.
Each student has to refer any one of the web sites stated above.

21.Final Notes

Web development refers to the creating, building, and maintaining of websites. It includes aspects such as web design, web publishing, web programming. It is the creation of an application that works over the internet i.e. websites.

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)

- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- Open Challenge (https://www.openchallenge.org/)

Student must have any one of the following certifications:

- Udemy- Web Development Masterclass
- coursera – Web Design for Everybody: Basic of Web Development and Coding
- NPTEL – Introduction to Modern Application Development.

V. TEXT BOOKS:

1. Thomas A. Powell, "*The Complete Reference*", "HTML and CSS", 5th edition, 2017
2. Elisabeth Robson , Eric Freeman,"*Head First HTML and CSS: A Learner's Guide to Creating Standards-Based Web Pages*", 2nd edition, 2012
3. Adam Boduchand Roy Derks, "*React and React Native*: A Complete Hands-on Guide to Modern Web and Mobile Development with React.js", 3rd edition, 2020.

VI. REFERENCE BOOKS:

1. W Hans Bergsten, "*Java Server Pages*", O' Reilly, 3rd edition, 2003.
2. D. Flanagan, "*Java Script*", O'Reilly, 6th edition, 2011.
3. Jon Duckett, "*Beginning Web Programming*", WROX, 2nd edition, 2008.

VII. ELECTRONICS RESOURCES:

1. https://www.codecademy.com/learn/paths/web-development/
2. https://nptel.ac.in/courses/106/105/106105084/
3. https://medium.com/@aureliomerenda/create-a-native-web-app-with-react-native-web419acac86b82
4. https://www.coursera.org/learn/react-native
5. https://desirecourse.net/react-native-and-redux-course-using-hooks

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

DATABASE MANAGEMENT SYSTEMS LABORATORY								
II Semester: CSE / CSE (AI & ML) / CSE (DS) / CSE (CS) / IT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AITD05	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

I. COURSE OVERVIEW:

The purpose of this course is to provide a clear understanding of fundamentals with emphasis on their applications to create and manage large data sets. It highlights on technical overview of database software to retrieve data from n database. The course includes database design principles, normalization, concurrent transaction processing, security, recovery and file organization techniques.

I. COURSES OBJECTIVES:

The students will try to learn

I. The SQL commands for data definition, manipulation, control and perform transactions in database systems.
II. The procedural language for implementation of functions, procedures, cursors and triggers using PL/SQL programs.
III. The logical design of a real time database system with the help of Entity Relationship diagrams.

II. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Demonstrate database creation and manipulation concepts with the help of SQL queries.
CO2	Make use of inbuilt functions of SQL queries to perform data aggregations, subqueries, embedded queries and views.
CO3	Apply key constraints on database for maintaining integrity and quality of data.
CO4	Demonstrate normalization techniques by using referential key constraints.
CO5	Implement PL/SQL programs on procedures, cursors and triggers for enhancing the features of database system to handle exceptions.
CO6	Design database model with the help of Entity Relationship diagrams for a real time system or scenario.

III. COURSE CONTENT:

EXERCISES FOR DATABASE MANAGEMENT SYSTEMS LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

Installation and Accessing Data base

Introduction:
A Database Management System (DBMS) is a software system that is designed to manage and organize data in a structured manner. It allows users to create, modify, and query a database, as well as manage the security and access controls for that database.
We can use different Database Languages and PL/SQL Programming for performing Database management in
Convenient manner.

.**Software:**
Oracle

Installation of Oracle Database:

Step 1: Go to Main Database Folder where you'll find Setup. Right click the setup.exe file and choose Run as Administrator.
Step 2: Click Yes to continue. This will start Oracle Universal Installer.
Step 3: Select any of the three different Installation Options according to your needs.
• Option 1 – If you want to Install Oracle Server Software and want to Create Database also.
• Option 2 – If you want to Install Oracle Server only.
• Option 3 – If you want to Upgrade your Existing Database.
Step 4: Choose between Server Class and Desktop Class as per your requirement and click on Next.
Step 5: Configure the basic settings and create a Password for your database. Once the configuration is done click on Next to continue.
Step 6: Oracle Universal Installer (OUI) will check for the Prerequisites such as Hardware compatibility.
Step 7: Click on Finish to start the Installation process. This installation might take some time depending on your Hardware.
Step 8: Click OK to finish the installation.
Step 9: Copy the localhost link provided to open your Enterprise Manager.
Step 10: Click the Close Button and you are done with the Installation Process.
Go to Start Menu and
• Search Oracle Folder
• Click on Database Control – Oracle (Your Global Database Name)
• This will take you to the Login Screen of your Oracle Enterprise Manager.

1.1 Creating and Deleting a Database - *CREATE DATABASE* and *DROP DATABASE*

You can create a new database using SQL command "`CREATE DATABASE` *`databaseName`*"; and delete a database using "`DROP DATABASE` *`databaseName`*". You could optionally apply condition "`IF EXISTS`" or "`IF NOT EXISTS`" to these commands. For example,

```
mysql> CREATE DATABASE southwind;
Query OK, 1 row affected (0.03 sec)

mysql> DROP DATABASE southwind;
Query OK, 0 rows affected (0.11 sec)

mysql> CREATE DATABASE IF NOT EXISTS southwind;
Query OK, 1 row affected (0.01 sec)

mysql> DROP DATABASE IF EXISTS southwind;
Query OK, 0 rows affected (0.00 sec)
```

IMPORTANT: Use SQL DROP (and DELETE) commands with extreme care, as the deleted entities are irrecoverable. **THERE IS NO UNDO!!!**

SHOW CREATE DATABASE

The CREATE DATABASE commands uses some defaults. You can issue a "SHOW CREATE DATABASE *databaseName*" to display the full command and check these default values. We use \G (instead of ';') to display the results vertically. (Try comparing the outputs produced by ';' and \G.)

```
mysql> CREATE DATABASE IF NOT EXISTS southwind;

mysql> SHOW CREATE DATABASE southwind \G
*************************** 1. row ***************************
       Database: southwind
Create Database: CREATE DATABASE `southwind` /*!40100 DEFAULT CHARACTER SET latin1 */
```

Try:
Create own database for storing all tables

2. Exercises on database definition language queries.

2.1 Create a table called Employee with the following structure.

Name	Type
Empno	Number
Ename	Varchar2(20)
Job	Varchar2(20)
Mgr	Number
Sal	Number

a. Add a column commission with domain to the Employee table.
b. Insert any five records into the table.
c. Update the column details of job

d. Rename the column of Employ table using alter command.
e. Delete the employee whose emp no is 19

```
create table employee (empno number, ename varchar2(10),job varchar2(10),mgr number,sal
number);
Table created.
SQL> desc employee;
Name Null Type
EMPNO NUMBER
ENAME VARCHAR2(10)
JOB VARCHAR2(10)
MGR NUMBER
SAL NUMBER
```

2.2 Add a column commission with domain to the Employee table.

```
SQL> alter table employee add(commission number); Table altered.
SQL> desc employee;
Name Null Type
EMPNO NUMBER
ENAME VARCHAR2(10)
JOB VARCHAR2(10)
MGR NUMBER
SAL NUMBER
COMMISSION NUMBE
```

2.3 Insert any five records into the table.

SQL> insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission');

```
Enter value for empno: 101
Enter value for ename: abhi Enter value for job: manager
Enter value for mgr: 1234 Enter value for sal: 10000 Enter value for commission: 70
Page 11
old 1: insert into employee values(&empno,'&ename','&job',&mgr,&sal,'&commission')
new 1: insert into employee values(101,'abhi','manager',1234,10000,'70')
1 row created.
```

2.4 Update the column details of job

SQL> update employee set job='trainee' where empno=103; 1 row updated.
SQL> select * from employee;

```
EMPNO ENAME JOB MGR SAL COMMISSION
101 abhi manager 1234 10000 70
102 rohith analyst 2345 9000 65
103 david trainee 3456 9000 65
104 rahul Clerk 4567 7000 55
105 pramod salesman 5678 5000 5
```

2.5 Rename the column of Employ table using alter command.

```
SQL> alter table employee rename column mgr to manager_no;
Table altered.
SQL> desc employee;
```

```
Name Null Type
EMPNO NUMBER
ENAME VARCHAR2(10)
JOB VARCHAR2(10)
MANAGER_NO NUMBER
SAL NUMBER
COMMISSION NUMBER
```

Try:

1. List the maximum salary paid to salesman
2. Display name of emp whose name start with "I‟
3. List details of emp who have joined before ‟30-sept-81‟
4. Display the emp details in the descending order of their basic salary
5. List of no of emp & avg salary for emp in the dept no „20‟

Hint:

Create a database called company consist of the following tables by using data definition languages.

1.Emp (eno, ename, job, hire date, salary, commission, deptno,)
2.dept (deptno, deptname, location)
eno is primary key in emp
deptno is primary key in dept

3. Exercises on database Manipulation language Queries.

3.1 Create a table called department with the following structure.

Name	Type
Deptno	Number
Deptna me location	Varchar2(2 0) Varchar2(2 0)

```
SQL> create table department (deptno number, deptname varchar2(10), location
varchar2(10));
```

```
Table created.
SQL> desc department;
DEPTNO NUMBER
DEPTNAME VARCHAR2(10)
LOCATION VARCHAR2(10)
```

3.2 Add column designation to the department table.

```
SQL> alter table department add(designation varchar2(10));
Table altered.
```

```
SQL> desc department;
DEPTNO NUMBER
DEPTNAME VARCHAR2(10)
LOCATION VARCHAR2(10)
DESIGNATION VARCHAR2(10)
```

3.3 Insert values into the table.

```
SQL> insert into department values(&deptno,’&deptname’,’&location’,&designation’);
```

```
Enter value for deptno: 9
Enter value for deptname: accounting Enter value for location: hyderabad Enter value for designation:
manager
```

```
old 1: insert into department values(&deptno,’&deptname’,’&location’,’&designation’)
new 1: insert into department values(9,’accounting’,’hyderabad’,’manager’)
```

```
1 row created.
SQL> /
```

```
Enter value for deptno: 10
Enter value for deptname: research Enter value for location: chennai Enter value for
designation:
professor
```

```
old 1: insert into department values(&deptno,’&deptname’,’&location’,’&designation’)
new 1: insert into department values(10,’research’,’chennai’,’professor’)
1 row created
```

3.4 List the records of emp table grouped by deptno.

```
SQL> select deptno,deptname from department group by deptno,deptname;
```

```
Page 13
DEPTNO DEPTNAME
9 accounting
12 operations
10 research
11 sales
```

3.5 Update the record designation where deptno is 17.

```
SQL> update department set designation=’accountant’ where deptno=17;
2 rows updated.
```

Try:

1. List full details of all Rooms.
2. How many Rooms are there.

3. List the price and type of all rooms at the Amrutha Hotel.
4. List the number of guests in each Room
5. Update the price of all rooms by 5%.

Hint:
The following tables form part of a database held in a relational DBMS by using data manipulation languages:

Hotel (HotelNo, Name, City) HotelNo is the primary key
Room (RoomNo, HotelNo, Type, Price)
Booking (HotelNo, GuestNo, DateFrom, DateTo, RoomNo)
Guest (GuestNo, GuestName, GuestAddress) GuestNo is primary key
Room contains room details for each hotel and (HotelNo, RoomNo) forms the primary key.
Booking contains details of the bookings and the primary key comprises (HotelNo, GuestNo and DateFrom)

4. Exercises on both DDL and DML Queries.

4.1 Create a table called Customer with the following structure.

Students can execute fallowing queries based on previous queries executed

Try:
1. Retrieve city, phone, url of author whose name is „CHETAN BHAGAT‟.
2. Retrieve book title, reviewable id and rating of all books.
3. Retrieve book title, price, author name and url for publishers „MEHTA‟.
4. In a PUBLISHER relation change the phone number of „MEHTA‟ to 123456
5. Calculate and display the average, maximum, minimum price of each publisher.
6. Delete details of all books having a page count less than 100.
7. Retrieve details of all authors residing in city Pune and whose name begins with character „C‟.
8. Retrieve details of authors residing in same city as „Korth‟.
9. Create a procedure to update the value of page count of a book of given ISBN.
10. Create a function that returns the price of book with a given ISBN.

Hint:
Create the following tables using data definition languages, data manipulation languages.
1)PUBLISHER (PID , PNAME ,ADDRESS ,STATE ,PHONE ,EMAILID);
2)BOOK (ISBN ,BOOK_TITLE , CATEGORY , PRICE , COPYRIGHT_DATE , YEAR ,PAGE_COUNT ,PID);
3) AUTHOR (AID,ANAME,STATE,CITY ,ZIP,PHONE,URL)
4) AUTHOR_BOOK (AID, ISBN);
5) REVIEW (RID, ISBN, RATING);

5. Exercises on tables with different types of constraints.

5.1 Not Null constraints.

```
CREATE TABLE STUDENT(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> insert into student values(6201 ,'raj',18,'meerpet');
```

```
Query OK, 1 row affected (0.02 sec)
```

```
mysql> insert into student values(NULL,'raj',18,'meerpet');
```

```
ERROR 1048 (23000): Column 'ROLL_NO' cannot be null
```

5.2 Unique constraints.

```
CREATE TABLE STUDENT1(
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (35) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> insert into student1 values(6201,'raj',18,'meerpet');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into student1 values(6201,'raj',18,'meerpet');
ERROR 1062 (23000): Duplicate entry '6201' for key 'PRIMARY'
mysql> insert into student1 values(6202,'raj',18,'meerpet');
ERROR 1062 (23000): Duplicate entry 'raj' for key 'STU_NAME'
mysql> insert into student1 values(6202,'ram',18,'meerpet');
ERROR 1062 (23000): Duplicate entry 'meerpet' for key 'STU_ADDRESS'
```

5.3 Default constraints.

```
CREATE TABLE STUDENT2(
ROLL_NO   INT  NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT  DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);
insert into
student2(ROLL_NO,STU_NAME,STU_AGE,STU_ADDRESS)values(6202,'sam',21,'lbnagar')
;
Query OK, 1 row affected (0.00 sec)

mysql>  select * from student2;
```

```
+---------+----------+---------+----------+-------------+
| ROLL_NO | STU_NAME | STU_AGE | EXAM_FEE | STU_ADDRESS |
+---------+----------+---------+----------+-------------+
|    6201 | raj      |      18 |    12000 | meerpet     |
|    6202 | sam      |      21 |    10000 | lbnagar     |
+---------+----------+---------+----------+-------------+
```

```
2 rows in set (0.00 sec)
```

5.4 Check constraints.

```
CREATE TABLE STUDENT3(
ROLL_NO   INT  NOT NULL CHECK(ROLL_NO >1000) ,
STU_NAME VARCHAR (35)  NOT NULL,
STU_AGE INT  NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (35) ,
PRIMARY KEY (ROLL_NO)
);

 insert into student3 values(112,'ramana',23,12000,'hyderabad');
Query OK, 1 row affected (0.01 sec)

mysql> select * from student3;
```

```
+---------+----------+---------+----------+-------------+
| ROLL_NO | STU_NAME | STU_AGE | EXAM_FEE | STU_ADDRESS |
+---------+----------+---------+----------+-------------+
|     112 | ramana   |      23 |    12000 | hyderabad   |
|     800 | raj      |      13 |    12000 | lbnagar     |
|     900 | raj      |      13 |    12000 | lbnagar     |
+---------+----------+---------+----------+-------------+
3 rows in set (0.00 sec)
```

5.5 Auto Increment Constraints.

```
CREATE TABLE Persons1 (
   Personid int NOT NULL AUTO_INCREMENT,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Age int,
   PRIMARY KEY (Personid)
);

INSERT INTO persons1 values(11,'reddy','ramana',23);
Query OK, 1 row affected (0.00 sec)

INSERT INTO persons1(LastName,FirstName,Age) values('reddy','ramana',23);
Query OK, 1 row affected (0.02 sec)

mysql> select * from persons1;
```

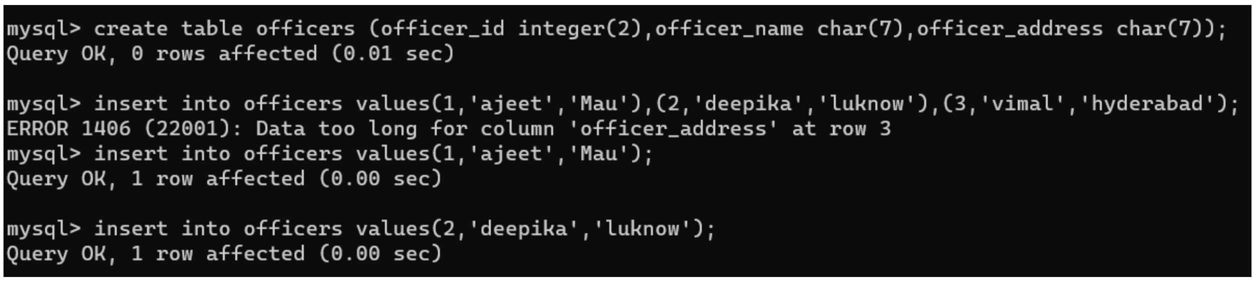
```
+----------+----------+-----------+------+
| Personid | LastName | FirstName | Age  |
+----------+----------+-----------+------+
|       11 | reddy    | ramana    |   23 |
|       12 | reddy    | ramana    |   23 |
+----------+----------+-----------+------+
2 rows in set (0.00 sec)
```

Try:
Create tables with both Primary key and foreign key constraints

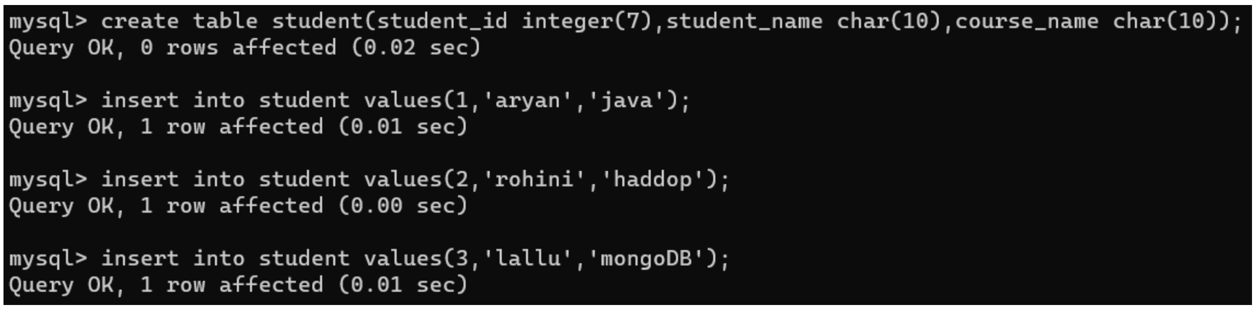
Hint:
Practice all constraints for obtaining syntax on both primary and foreign key constraints that applied on ID.

6. Exercises on Join Operations and Aggregate Functions in Database.

6.1 Create table names called 'students' and 'officers' and insert values as show in below.



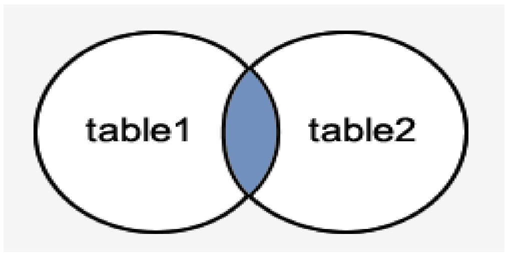
```
mysql> create table officers (officer_id integer(2),officer_name char(7),officer_address char(7));
Query OK, 0 rows affected (0.01 sec)

mysql> insert into officers values(1,'ajeet','Mau'),(2,'deepika','luknow'),(3,'vimal','hyderabad');
ERROR 1406 (22001): Data too long for column 'officer_address' at row 3
mysql> insert into officers values(1,'ajeet','Mau');
Query OK, 1 row affected (0.00 sec)

mysql> insert into officers values(2,'deepika','luknow');
Query OK, 1 row affected (0.00 sec)
```



```
mysql> create table student(student_id integer(7),student_name char(10),course_name char(10));
Query OK, 0 rows affected (0.02 sec)

mysql> insert into student values(1,'aryan','java');
Query OK, 1 row affected (0.01 sec)

mysql> insert into student values(2,'rohini','haddop');
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(3,'lallu','mongoDB');
Query OK, 1 row affected (0.01 sec)
```

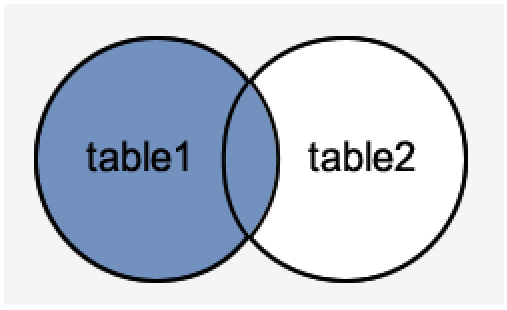
6.2 Inner join

1. SELECT officers.officer_name, officers.officer_address, students.course_name
2. FROM officers
3. INNER JOIN students
4. ON officers.officer_id = students.student_id;



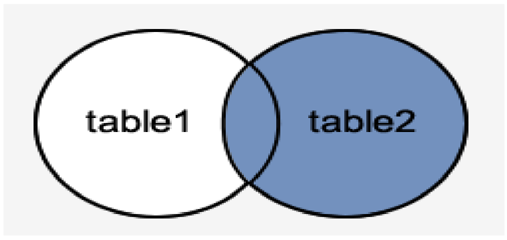
6.3 Left Outer Join

1. SELECT officers.officer_name, officers.officer_address, students.course_name
2. FROM officers
3. LEFT JOIN students
4. ON officers.officer_id = students.student_id;



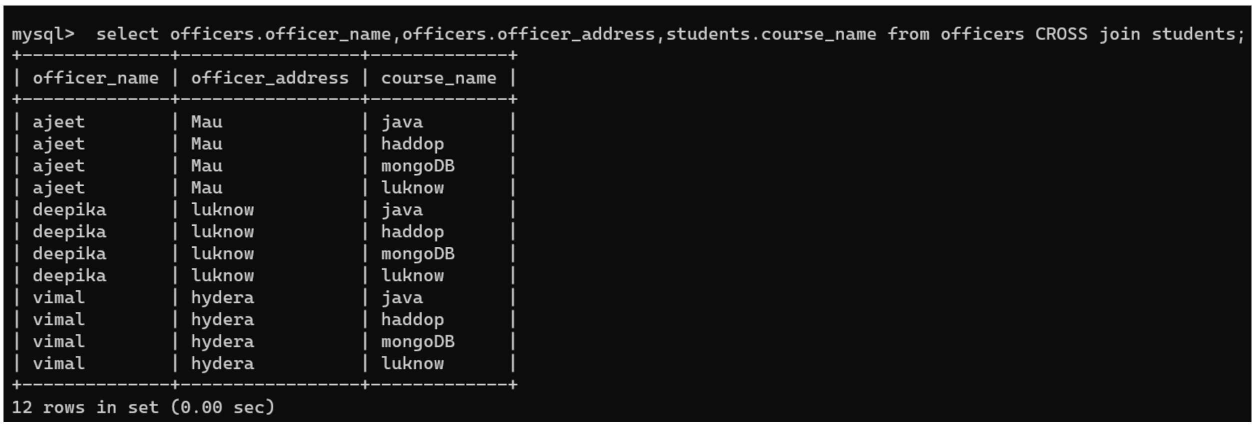
6.4 Right Outer Join

1. SELECT officers.officer_name, officers.officer_address, students.course_name, students.student_name
2. FROM officers
3. RIGHT JOIN students
4. ON officers.officer_id = students.student_id;



6.5 Cross Join

SSelect officers.officer_name,officers.officer_address,students.course_name from officers CROSS join students;



```
mysql>  select officers.officer_name,officers.officer_address,students.course_name from officers CROSS join students;
+--------------+-----------------+-------------+
| officer_name | officer_address | course_name |
+--------------+-----------------+-------------+
| ajeet        | Mau             | java        |
| ajeet        | Mau             | haddop      |
| ajeet        | Mau             | mongoDB     |
| ajeet        | Mau             | luknow      |
| deepika      | luknow          | java        |
| deepika      | luknow          | haddop      |
| deepika      | luknow          | mongoDB     |
| deepika      | luknow          | luknow      |
| vimal        | hydera          | java        |
| vimal        | hydera          | haddop      |
| vimal        | hydera          | mongoDB     |
| vimal        | hydera          | luknow      |
+--------------+-----------------+-------------+
12 rows in set (0.00 sec)
```

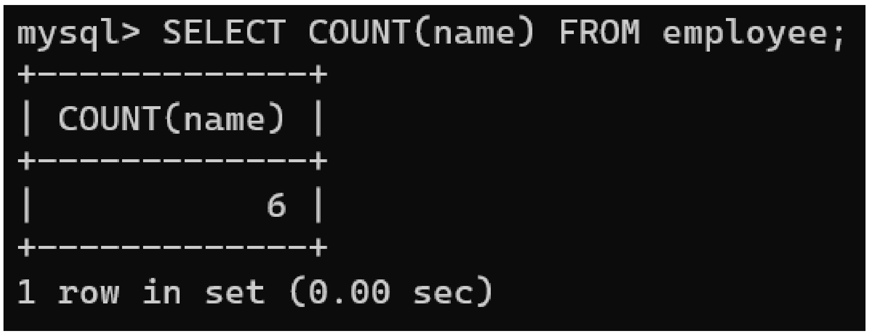
Try:
Perform Self join, Equi join, Natural joins on tables to get the values from multiple tables.

Hint:
Understand and use same syntax applied for all joins can use as same it is for Self-join, Equi join, Natural joins.

6.6 AGGREGATE FUNCTIONS:

1. Count() Function:

```
SELECT COUNT(name) FROM employee;
```



```
mysql> SELECT COUNT(name) FROM employee;
+-------------+
| COUNT(name) |
+-------------+
|           6 |
+-------------+
1 row in set (0.00 sec)
```

2. Sum() Function

```
SELECT SUM(working_hours) AS "Total working hours" FROM employee;
```

3. AVG() Function:

```
SELECT AVG(working_hours) AS "Average working hours" FROM employee;
```

4. MIN() Function:

```
SELECT MIN(working_hours) AS Minimum_working_hours FROM employee;
```

5. MAX() Function:

```
SELECT MAX(working_hours) AS Maximum_working_hours FROM employee;
```

Try:
Execute other aggregate functions to find first and Last employees

Hint:
Use First and Last functions as aggregate functions to execute

7. Exercises on TCL (transaction control language) Queries to perform rollbacking and save point mechanisms.

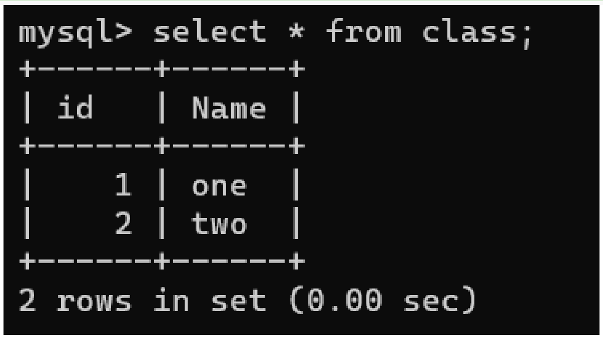
7.1 Create table names called 'class' and insert values into table.

```
mysql> create table class (id int(5),Name char(10));
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> insert into class values(1,'one'),(2,'two');
```

```
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```



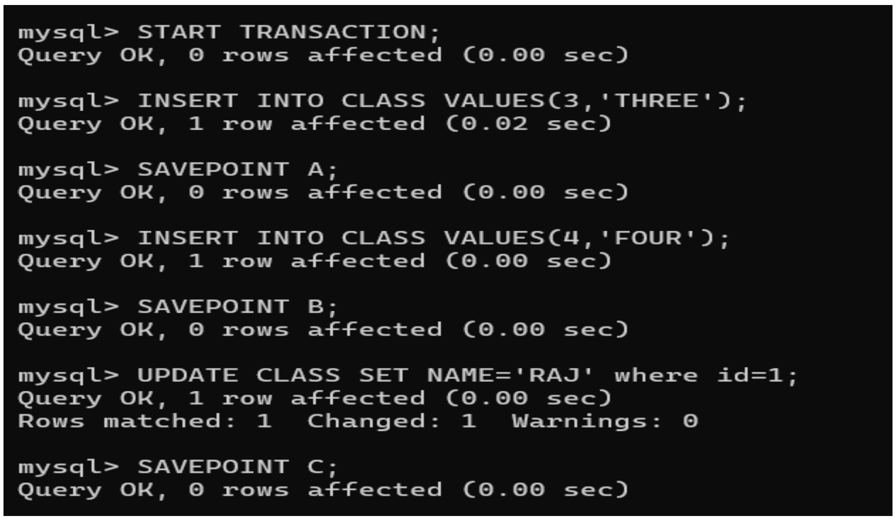
```
mysql> select * from class;
+------+------+
| id   | Name |
+------+------+
|    1 | one  |
|    2 | two  |
+------+------+
2 rows in set (0.00 sec)
```

Now to perform TCL we need start the transaction:

For that START TRANSACTION is a command to do start our transaction.
SAVEPOINT works only before commit operation.
After commit operation SAVEPOINT will not work.

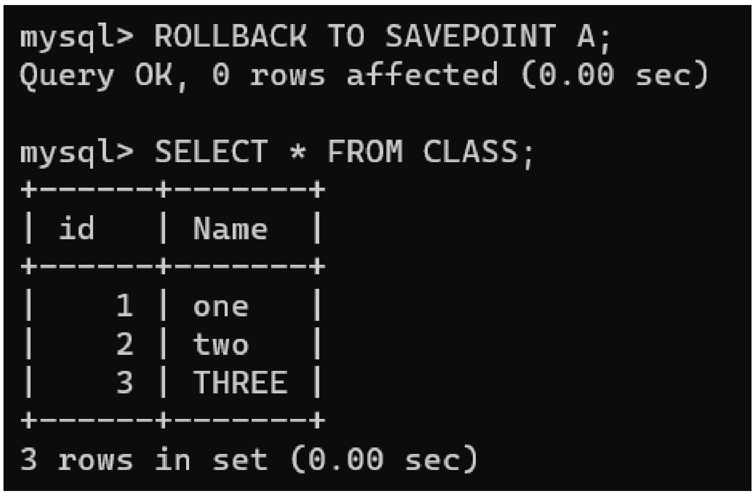
So save point saves the current work and displays the information.

7.2 Before commit performs



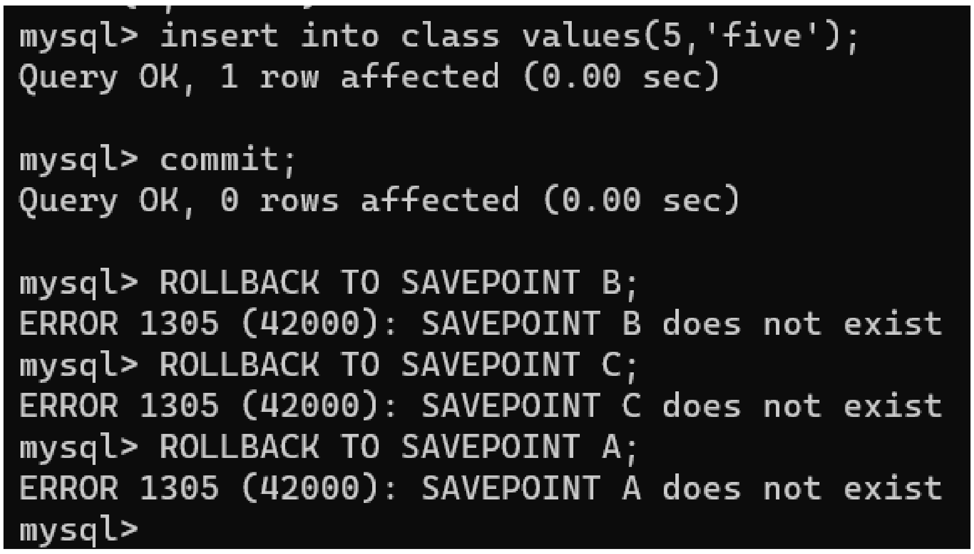
```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO CLASS VALUES(3,'THREE');
Query OK, 1 row affected (0.02 sec)

mysql> SAVEPOINT A;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO CLASS VALUES(4,'FOUR');
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT B;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE CLASS SET NAME='RAJ' where id=1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SAVEPOINT C;
Query OK, 0 rows affected (0.00 sec)
```

NOW
If we perform COMMIT operation then we are not rollback the data which happens on save point states;
We can roll back the data which doesn't performed with commit operation.



```
mysql> ROLLBACK TO SAVEPOINT A;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM CLASS;
+------+-------+
| id   | Name  |
+------+-------+
|    1 | one   |
|    2 | two   |
|    3 | THREE |
+------+-------+
3 rows in set (0.00 sec)
```

.3 After commit performs



```
mysql> insert into class values(5,'five');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> ROLLBACK TO SAVEPOINT B;
ERROR 1305 (42000): SAVEPOINT B does not exist
mysql> ROLLBACK TO SAVEPOINT C;
ERROR 1305 (42000): SAVEPOINT C does not exist
mysql> ROLLBACK TO SAVEPOINT A;
ERROR 1305 (42000): SAVEPOINT A does not exist
mysql>
```

So after COMMIT No information will rollback.
All the above commit,
rollback, save point operations can be done after START TRANSACTION only.

Try:
Insert two more values and perform commit operation then roll back again

Hint:

Perform before commit operation to rollback data from various operations performed

8. Exercises on user accounts to grant privileges using Data Control language process.

8.1 Create user.

create user ‘username’@‘localhost’ IDENTIFIED BY ‘password’;
Ex:

create user 'iare' identified by 'iare';
Query OK, 0 rows affected (0.05 sec)

SEEL LIST OF USERS IN DATABASE:

select user from mysql.user;

TO SEE CURRENT USER:

select current_user();

TO SEE DESCRIPTION OF USER:

desc mysql.user;

TO SEE LIST OF USERS, HOST:

```
select user,host from mysql.user;
```

8.2 To Grant All Privileges, Specific Privileges to The User:

SYNTAX:

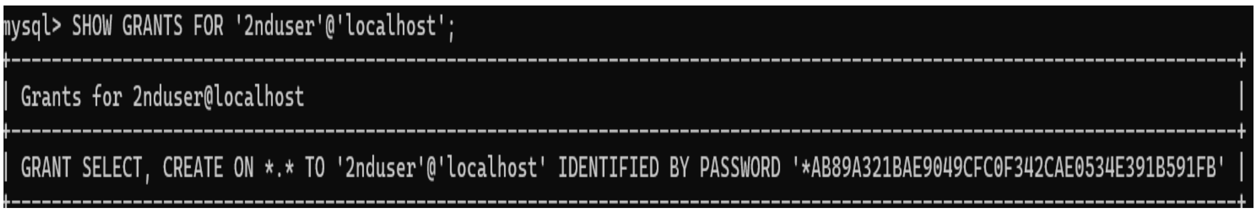
```
GRANT ALL PRIVILEGES ON * . * TO 'new_user'@'localhost';
GRANT ALL PRIVILEGES ON * . * TO ‘2nduser’@’localhost’;
Query OK, 0 rows affected (0.00 sec)
```

TO GRANT SPECIFIC PRIVILEGES TO THE USER:

```
GRANT CREATE, SELECT ON * . * TO '2nduser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

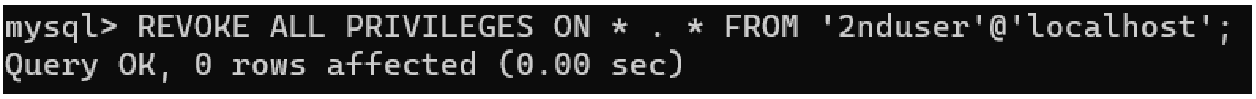
SHOW PRIVILEGES TO THE USER:

```
SHOW GRANTS FOR '2nduser'@'localhost';
```



```
mysql> SHOW GRANTS FOR '2nduser'@'localhost';
+------------------------------------------------------------------------------------------------------------+
| Grants for 2nduser@localhost                                                                               |
+------------------------------------------------------------------------------------------------------------+
| GRANT SELECT, CREATE ON *.* TO '2nduser'@'localhost' IDENTIFIED BY PASSWORD '*AB89A321BAE9049CFC0F342CAE0534E391B591FB' |
+------------------------------------------------------------------------------------------------------------+
```

8.3 Revoke permissions from user:

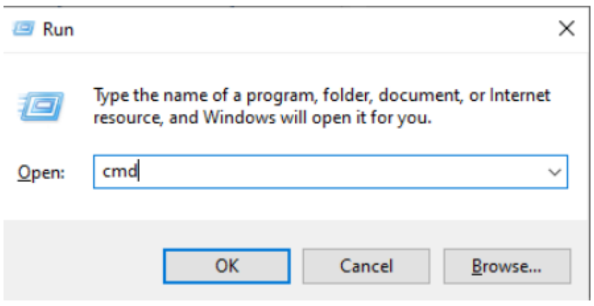


```
mysql> REVOKE ALL PRIVILEGES ON * . * FROM '2nduser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

8.4 Login as User:

Before login as a user, the user has to privileged with root user as mentioned above grants option. That means a new user having permission to login as user.

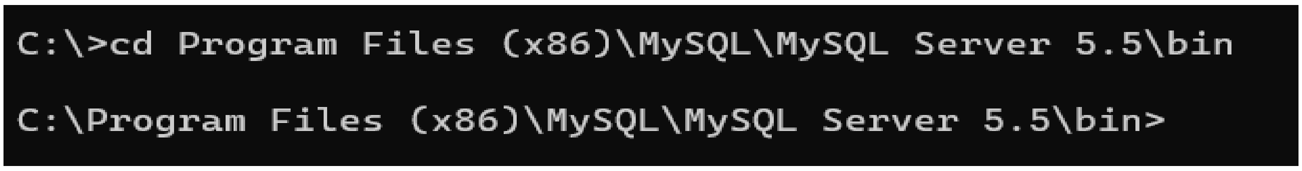
Now
To login to a different user account, we need to open the command prompt by executing the run command.


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: cmd
OK
Cancel
Browse...

After clicking the **OK** button, we can see the command prompt
Now move to the path where my sql has been specified

C drive->program files(x86)->mysql->mysql server5.5->bin

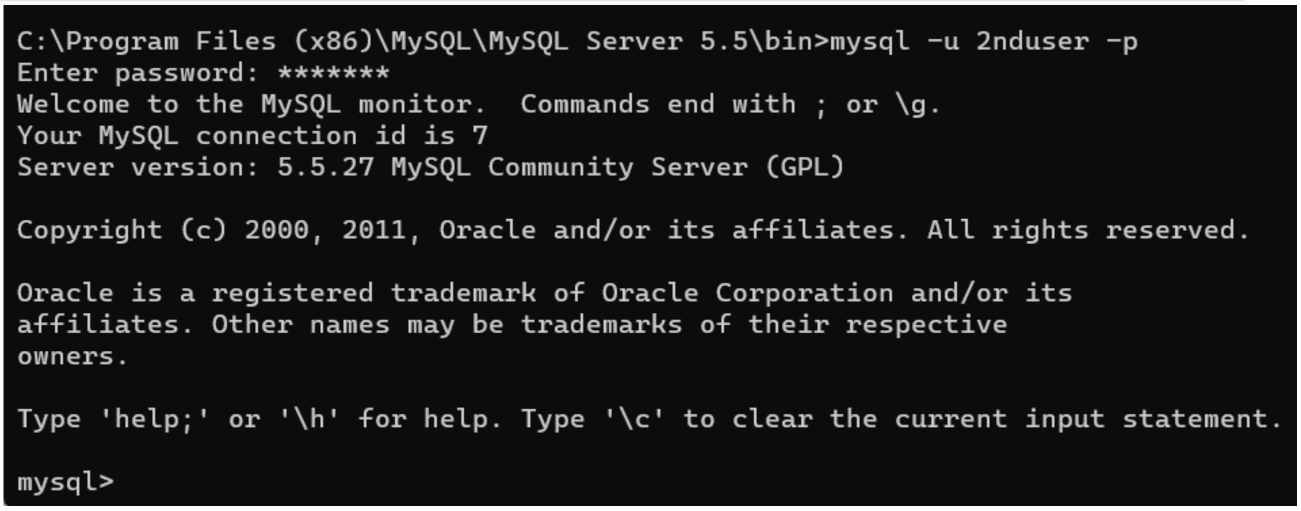
As shown in below



```
C:\>cd Program Files (x86)\MySQL\MySQL Server 5.5\bin

C:\Program Files (x86)\MySQL\MySQL Server 5.5\bin>
```

Next type the fallowing syntax to login as user:

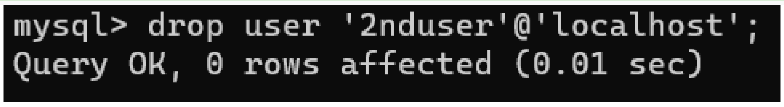
```
Mysql -u username -p
```



```
C:\Program Files (x86)\MySQL\MySQL Server 5.5\bin>mysql -u 2nduser -p
Enter password: *******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.5.27 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

8.5 DROP USERS FROM DATABASE:

```
DROP USER 'user_name'@'localhost';
```



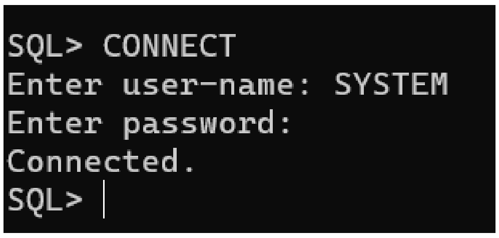
```
mysql> drop user '2nduser'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Try:
create any other user and grant specific privileges on create and select permissions

Hint:
create user and perform DCL operations to give privileges and revoke privileges

9. Working with basic PL/SQL programing Examples

To start with pl/sql programming
Open oracle run sql cmd line avialable in oracle
After opening window
Just type CONNECT and hit enter
Now
 it is asking you to enter user-name and password;
Enter both and now it is connected as shown in below



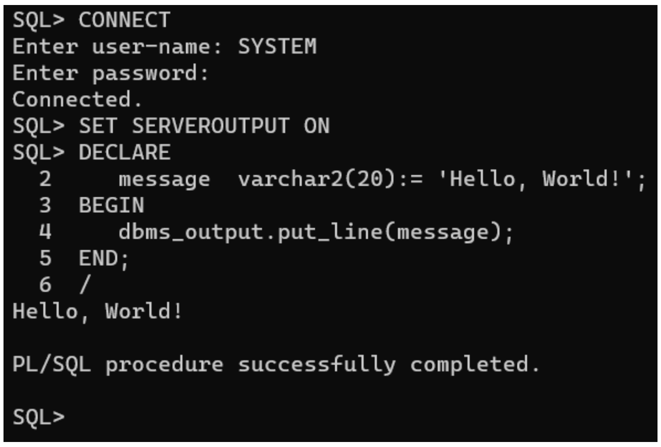
```
SQL> CONNECT
Enter user-name: SYSTEM
Enter password:
Connected.
SQL>
```

```
SQL> CONNECT
Enter user-name: SYSTEM
Enter password: admin
Connected.
```

Before starting pl/sql programming we need to set server;
To start server type fallowing command after connected as shown in above.
SET SERVEROUTPUT ON

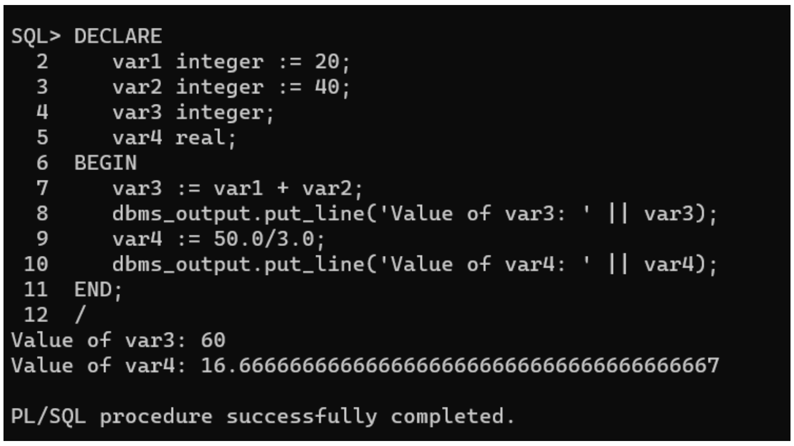
```
After that we can start to write program as per the below syntax
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling>
END;
```

9.1 Hello world example with programming



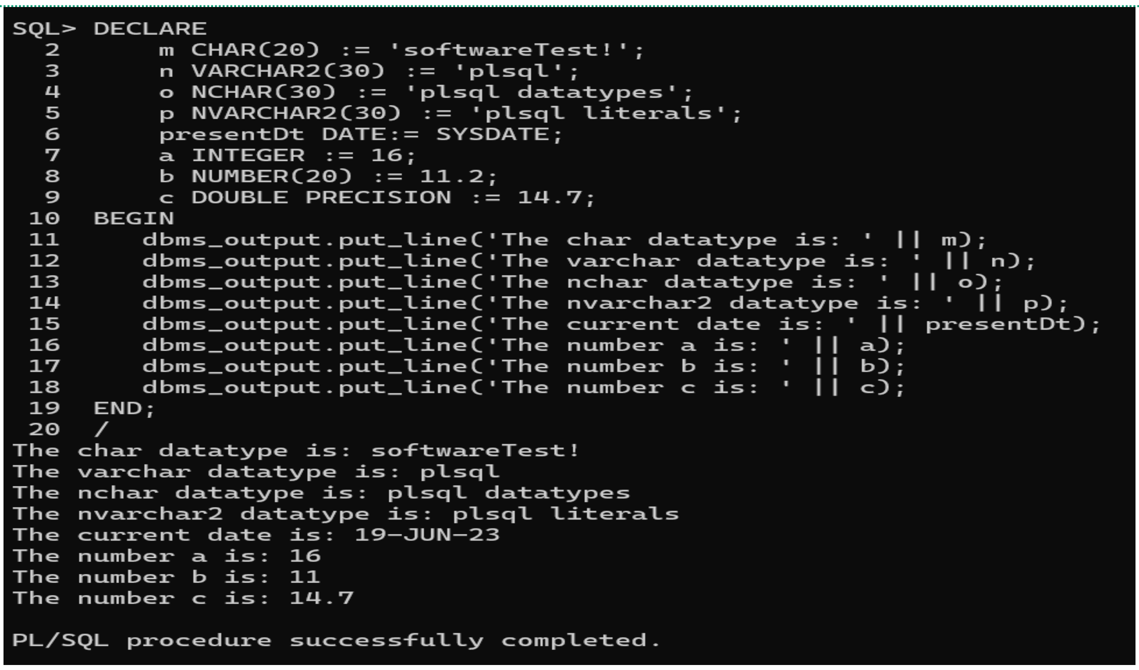
```
SQL> CONNECT
Enter user-name: SYSTEM
Enter password:
Connected.
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     message  varchar2(20):= 'Hello, World!';
  3  BEGIN
  4     dbms_output.put_line(message);
  5  END;
  6  /
Hello, World!

PL/SQL procedure successfully completed.

SQL>
```

9.2 Declaration and Initialization of Variables



```
SQL> DECLARE
  2     var1 integer := 20;
  3     var2 integer := 40;
  4     var3 integer;
  5     var4 real;
  6  BEGIN
  7     var3 := var1 + var2;
  8     dbms_output.put_line('Value of var3: ' || var3);
  9     var4 := 50.0/3.0;
 10     dbms_output.put_line('Value of var4: ' || var4);
 11  END;
 12  /
Value of var3: 60
Value of var4: 16.66666666666666666666666666666666666667

PL/SQL procedure successfully completed.
```

9.3 Datatypes and its values



```
SQL> DECLARE
  2       m CHAR(20) := 'softwareTest!';
  3       n VARCHAR2(30) := 'plsql';
  4       o NCHAR(30) := 'plsql datatypes';
  5       p NVARCHAR2(30) := 'plsql literals';
  6       presentDt DATE:= SYSDATE;
  7       a INTEGER := 16;
  8       b NUMBER(20) := 11.2;
  9       c DOUBLE PRECISION := 14.7;
 10  BEGIN
 11     dbms_output.put_line('The char datatype is: ' || m);
 12     dbms_output.put_line('The varchar datatype is: ' || n);
 13     dbms_output.put_line('The nchar datatype is: ' || o);
 14     dbms_output.put_line('The nvarchar2 datatype is: ' || p);
 15     dbms_output.put_line('The current date is: ' || presentDt);
 16     dbms_output.put_line('The number a is: ' || a);
 17     dbms_output.put_line('The number b is: ' || b);
 18     dbms_output.put_line('The number c is: ' || c);
 19  END;
 20  /
The char datatype is: softwareTest!
The varchar datatype is: plsql
The nchar datatype is: plsql datatypes
The nvarchar2 datatype is: plsql literals
The current date is: 19-JUN-23
The number a is: 16
The number b is: 11
The number c is: 14.7

PL/SQL procedure successfully completed.
```

Try:
write program to display both local and global variable values
Hint:
use the concept local and global declaration and its variables

10. Working with PL/SQL Programming Functions.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
   < function_body >
END [function_name];
```

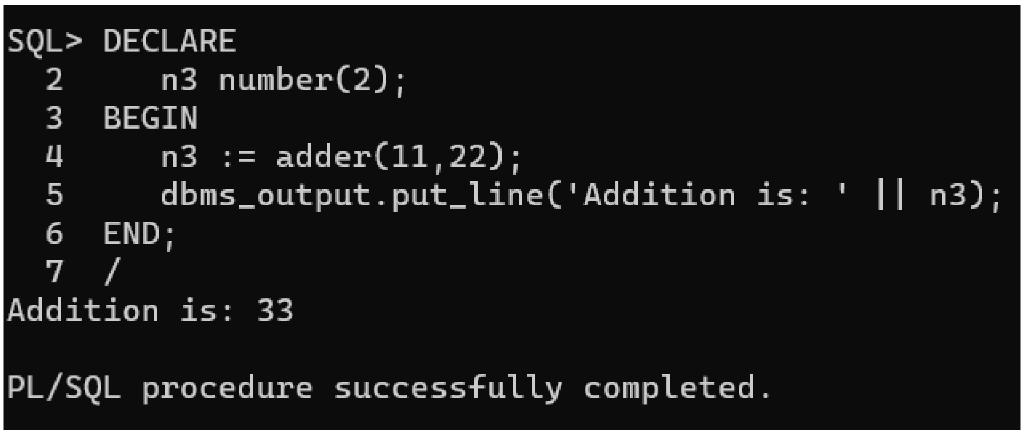
10.1 Addition of two numbers

Creating function:

```
create or replace function adder(n1 in number, n2 in number)
return number
is
n3 number(8);
begin
n3 :=n1+n2;
return n3;
end;
/
```

call the function.:

```
DECLARE
   n3 number(2);
BEGIN
   n3 := adder(11,22);
   dbms_output.put_line('Addition is: ' || n3);
END;
/
```



```
SQL> DECLARE
  2      n3 number(2);
  3   BEGIN
  4      n3 := adder(11,22);
  5      dbms_output.put_line('Addition is: ' || n3);
  6   END;
  7   /
Addition is: 33

PL/SQL procedure successfully completed.
```

10.2 compute and return the maximum of two values.

```
DECLARE
   a number;
   b number;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      Z:= y;
   END IF;
  RETURN z;
END;
BEGIN
   a:= 23;
   b:= 45;
   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

10.3 creating and calling function to access values from table.

To do this first create a table as shown in below

```
CREATE TABLE customers1
   ( customer_id number(10) NOT NULL,
   customer_name varchar2(50) NOT NULL,
  city varchar2(50)
  );
```

Next insert values into table as shown in below

```
insert into customers1 values(12,'raj','hyderabad');

insert into customers1 values(13,'ram','secunderabad');
```

Now create a function to count the no of customers in table:

Creating function:

```
CREATE OR REPLACE FUNCTION Fcustomers
RETURN number IS
   total number(2) := 0;
BEGIN
   SELECT count(*) into total
   FROM customers1;
    RETURN total;
END;
/
```

Calling function:

```
DECLARE
   c number(2);
BEGIN
   c := Fcustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/.
```

Try:
Create a function that updates the value of any column in the table.

Hint:
To update we use dml commands and call function to access values from table.

11. Working with PL/SQL Programming procedures.

Syntax:

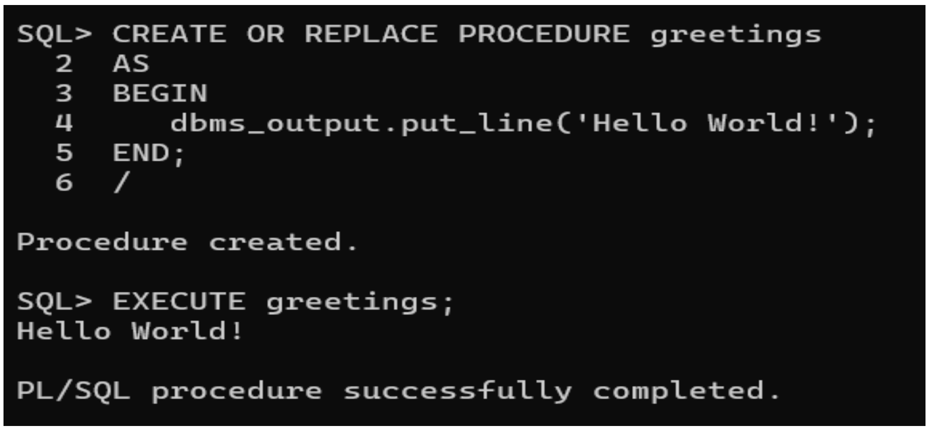
```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
  < procedure_body >
END procedure_name;
```

11.1 Create procedure to display hello world.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
/
```

TO EXECUTE:

```
EXECUTE greetings;
```



```
SQL> CREATE OR REPLACE PROCEDURE greetings
  2  AS
  3  BEGIN
  4     dbms_output.put_line('Hello World!');
  5  END;
  6  /

Procedure created.

SQL> EXECUTE greetings;
Hello World!

PL/SQL procedure successfully completed.
```

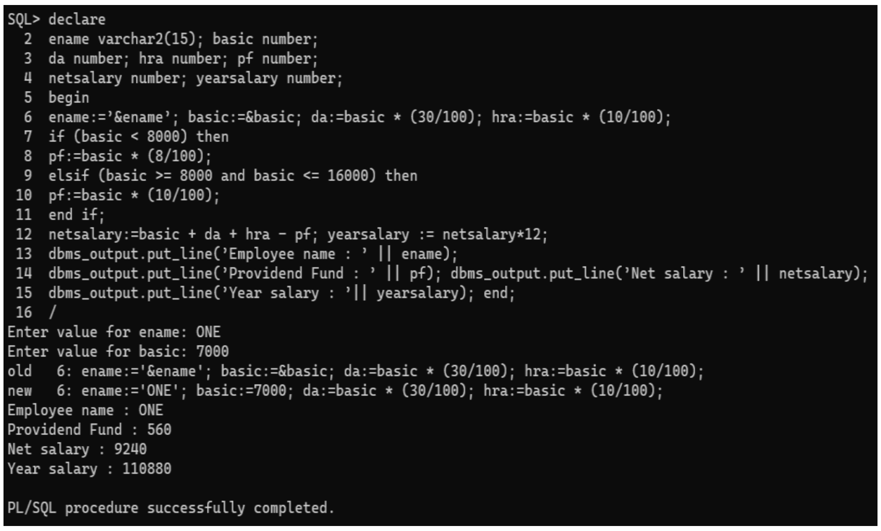
11.2 IN & OUT Mode Example to display square of number

```
DECLARE
   a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
  x := x * x;
END;
BEGIN
   a:= 23;
   squareNum(a);
   dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

11.3 calculate the net salary and year salary if da is 30% of basic, hra is 10% of basic and pf is 7% if basic salary is less than 8000, pf is 10% if basic sal between 8000 to 160000.

```
declare
ename varchar2(15); basic number;
da number; hra number; pf number;
```

```
netsalary number; yearsalary number;
begin
ename:='&ename'; basic:=&basic; da:=basic * (30/100); hra:=basic * (10/100);
if (basic < 8000) then
pf:=basic * (8/100);
elsif (basic >= 8000 and basic <= 16000) then
pf:=basic * (10/100);
end if;
netsalary:=basic + da + hra - pf; yearsalary := netsalary*12;
dbms_output.put_line('Employee name : ' || ename);
dbms_output.put_line('Providend Fund : ' || pf); dbms_output.put_line('Net salary : ' || netsalary);
dbms_output.put_line('Year salary : '|| yearsalary); end;
/
```



```
SQL> declare
  2  ename varchar2(15); basic number;
  3  da number; hra number; pf number;
  4  netsalary number; yearsalary number;
  5  begin
  6  ename:='&ename'; basic:=&basic; da:=basic * (30/100); hra:=basic * (10/100);
  7  if (basic < 8000) then
  8  pf:=basic * (8/100);
  9  elsif (basic >= 8000 and basic <= 16000) then
 10  pf:=basic * (10/100);
 11  end if;
 12  netsalary:=basic + da + hra - pf; yearsalary := netsalary*12;
 13  dbms_output.put_line('Employee name : ' || ename);
 14  dbms_output.put_line('Providend Fund : ' || pf); dbms_output.put_line('Net salary : ' || netsalary);
 15  dbms_output.put_line('Year salary : '|| yearsalary); end;
 16  /
Enter value for ename: ONE
Enter value for basic: 7000
old   6: ename:='&ename'; basic:=&basic; da:=basic * (30/100); hra:=basic * (10/100);
new   6: ename:='ONE'; basic:=7000; da:=basic * (30/100); hra:=basic * (10/100);
Employee name : ONE
Providend Fund : 560
Net salary : 9240
Year salary : 110880

PL/SQL procedure successfully completed.
```

Try:
Write procedure to find maximum and minimum of two numbers

Hint:
create and call procedure, use conditions to find max or minimum numbers

12. Working with PL/SQL Programming Triggers.

Triggers are stored programs, which are automatically executed or fired when some event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.

12.1 Syntax for creating trigger:

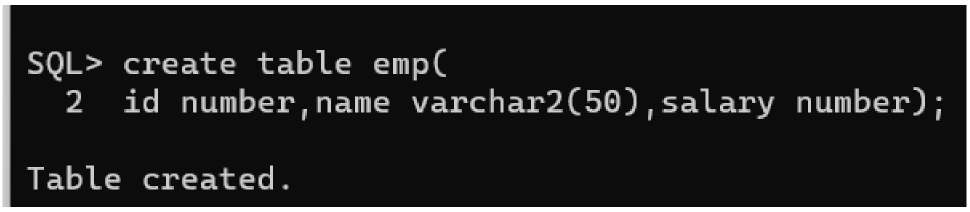
```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
```

WHEN (condition)

DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;

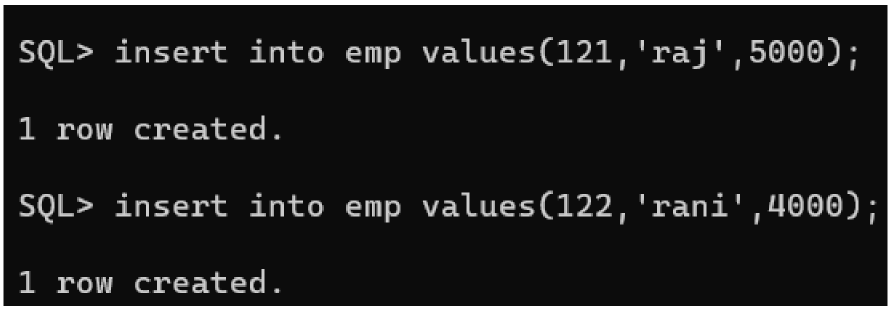
To do this first create a table called emp:

create table emp (id number, name varchar2(50),salary number);



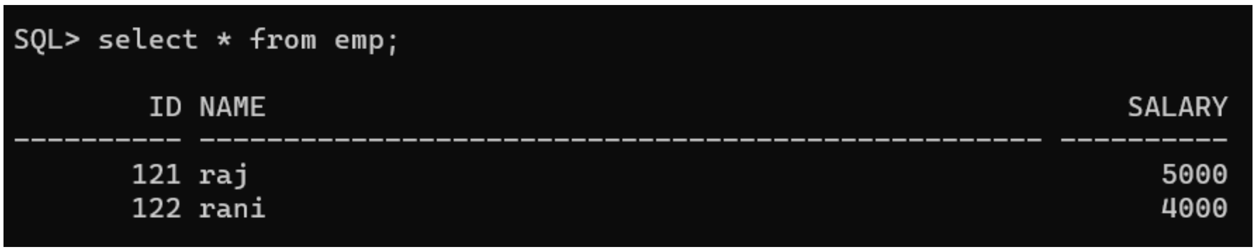
```
SQL> create table emp(
  2  id number,name varchar2(50),salary number);

Table created.
```

Insert any values:



```
SQL> insert into emp values(121,'raj',5000);

1 row created.

SQL> insert into emp values(122,'rani',4000);

1 row created.
```

Check values inserted or not:



```
SQL> select * from emp;

        ID NAME                                                   SALARY
---------- -------------------------------------------------- ----------
       121 raj                                                      5000
       122 rani                                                     4000
```

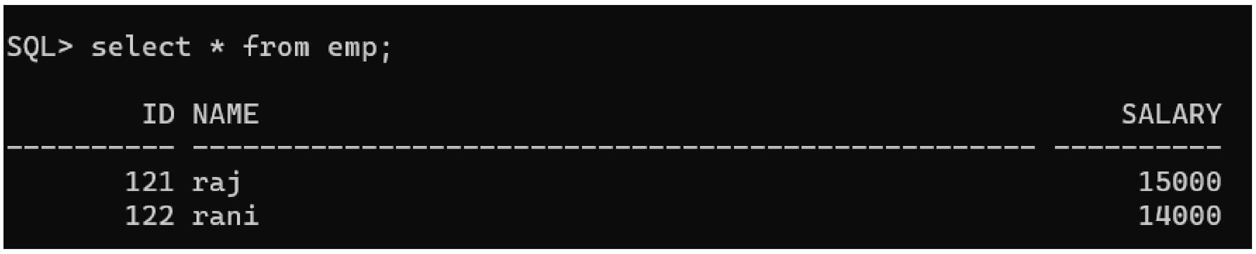
12.1 Creating the trigger for updating salary:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON emp
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

12.2 Calling the trigger for updating salary:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  emp
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

12.2 Check the salary is updated or not:



```
SQL> select * from emp;

        ID NAME                                               SALARY
---------- -------------------------------------------------- ----------
       121 raj                                                     15000
       122 rani                                                    14000
```

Try:
Create the trigger to delete an employee name ‘raj’

Hint:
Use DML and DDL commands, triggers to delete the employee’s name ‘raj’

13. Working with PL/SQL Programming Cursors.

A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.
A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

1. Implicit Cursors
2. Explicit Cursors

13.1 syntax for explicit cursor declaration

```
Declare the cursor:
Syntax for explicit cursor declaration
CURSOR name IS
 SELECT statement;
Open the cursor:
OPEN cursor_name;
Fetch the cursor:
FETCH cursor_name INTO variable_list;
Close the cursor:
Close cursor_name;
```

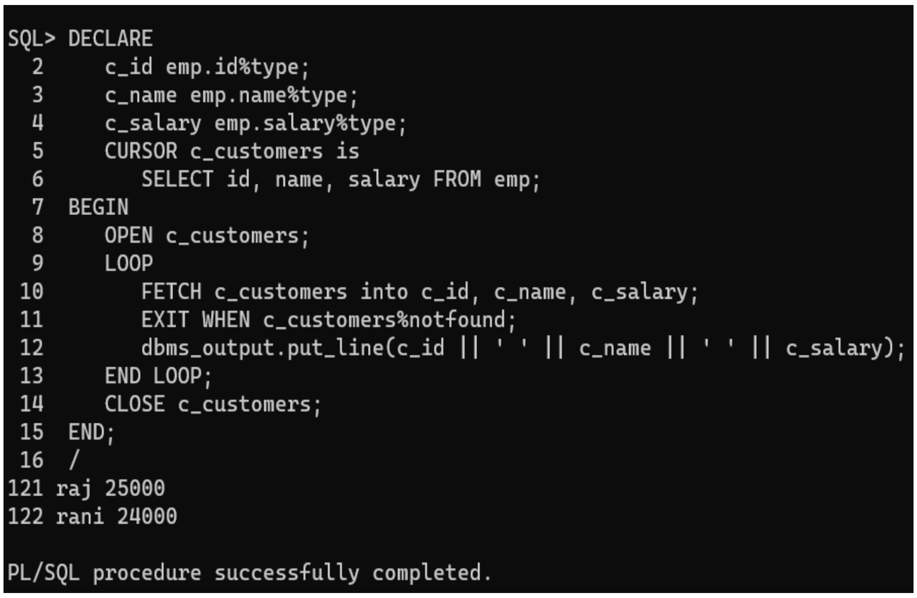
13.2 Implicit Cursor for customers salary update

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  emp
  SET salary = salary + 5000;
  IF sql%notfound THEN
     dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
     total_rows := sql%rowcount;
     dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

13.3 Explicit cursor for displaying updated salary

```
DECLARE
  c_id emp.id%type;
  c_name emp.name%type;
  c_salary emp.salary%type;
  CURSOR c_customers is
     SELECT id, name, salary FROM emp;
BEGIN
```

```
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_salary;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_salary);
  END LOOP;
  CLOSE c_customers;
END;
/
```



```
SQL> DECLARE
      c_id emp.id%type;
      c_name emp.name%type;
      c_salary emp.salary%type;
      CURSOR c_customers is
         SELECT id, name, salary FROM emp;
     BEGIN
      OPEN c_customers;
      LOOP
         FETCH c_customers into c_id, c_name, c_salary;
         EXIT WHEN c_customers%notfound;
         dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_salary);
      END LOOP;
      CLOSE c_customers;
     END;
     /
121 raj 25000
122 rani 24000

PL/SQL procedure successfully completed.
```

Try:
Create cursor to delete the employee whose salary is more than 24000

Hint:
Use DDL and DML commands with cursors to delete the employee.

V. REFERENCE BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, “Database System Concepts”, Mc raw-Hill, 4th edition,2002.
2. Ivan Bayross, “SQL, PL/SQL The programming language of oracle”, BPB publications, 4th Revised edition, 2010.

VI. WEB REFERENCE:

1 Ramez Elmasri, Shamkant, B. Navathe, “Database Systems”, Pearson Education, 6th edition, 2013.
2 Peter Rob, Carles Coronel, “Database System Concepts”, Cengage Learning, 7th edition, 2008.
3 M L Gillenson, “Introduction to Database Management”, Wiley Student edition,2012.

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

DevOps Engineering									
IV Semester: AERO | ME | CE | EEE | CSE | IT | CSE (AI&ML) | CSE (DS) | CSE (CS)									
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**			
ACSD18	Skill	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**	
		2	0	0	0	40	60	100	
Contact Classes: 32	**Tutorial Classes: Nil**	**Practical Classes: Nil**				**Total Classes: 32**			
Prerequisite: OBJECT ORIENTED PROGRAMMING									

I. COURSE OVERVIEW:
DevOps, a combination of "development" and "operations," is a software development methodology that emphasizes collaboration and communication between software developers and IT operations professionals. The goal of DevOps is to streamline the software delivery process, from code development to deployment and maintenance, by breaking down silos between development and operations teams.

II. COURSES OBJECTIVES:
The students will try to learn

I. The DevOps Concepts for business cases, cloud provisioning and management services .
II. The model canvas for DevOps use cases.
III. The virtual machines and containers for designing of applications
IV. The code with various aspects in continuous deployment / development.

III. COURSE OUTCOMES:
At the end of the course students should be able to:

CO 1	Understands the DevOps concepts in continuous delivery / development of applications.
CO 2	Create the DevOps applications using various tools and technologies.
CO 3	Examine the virtual machines and containers for managing the files
CO 4	Apply cloud services for deployment the applications in a real-time
CO 5	Perform web security and testing the code with appropriate tools

IV. COURSE SYLLABUS:

MODULE 1: DevOps Concepts
Understanding DevOps movement, DevOps with changing time, The water fall model, Agile Model, Collaboration, Why DevOps, Benefits of DevOps, DevOps life cycle- all about continuous, Build Automation, Continuous Integration, Continuous Management, Continuous Delivery / Continuous Development, The agile wheel of wheels.

MODULE 2: DevOps Tools and Technologies
Code Repositories : Git, Differences between SVN and Git, Build tools – Maven, Continuous integration tools – Jenkins, Container Technology – Docker, Monitoring Tools, Continuous integration with Jenkins 2, Creating built-in delivery pipelines, Creating Scripts, Creating a pipeline for compiling and executing test units, Using the Build Pipeline plugin, Integrating the deployment operation

MODULE 3: Docker Containers

Overview of Docker containers, Understanding the difference between virtual machines and containers, Installation and configuration of Docker, Creating your first Docker container, Managing containers, Creating a Docker image from Docker file, An overview of Docker's elements, Creating a Dockerfile, Writing a Dockerfile, Building and running a container on a local machine, Testing a container locally, Pushing an image to Docker Hub

MODULE 4: Cloud Provisioning and Configuration Management with Chef, Managing Containers Effectively with Kubernetes

Amazon EC2, Creating and configuring a virtual machine in Amazon Web Services, Prerequisite – deploying our application on a remote server, Deploying the application on AWS, Deploying the application in a Docker container.

Kubernetes architecture overview, Installing Kubernetes on a local machine, Installing the Kubernetes dashboard, Kubernetes application deployment, Using Azure Kubernetes Service (AKS), Creating an AKS service, Configuring kubectl for AKS, The build and push of the image in the Docker Hub

MODULE 5: Testing the Code

Manual testing, Unit testing, JUnit in general and JUnit in particular, A JUnit example, Automated integration testing, Docker in automated testing, Performance testing, Automated acceptance testing, Automated GUI testing, Integrating Selenium tests in Jenkins, JavaScript testing, Testing backend integration points, Test-driven development, A complete test automation scenario, Manually testing our web application.

V. TEXT BOOKS:

1. Mitesh Soni, "DevOps for Web Development", Packt Publishing, 2016.
2. Mikael Krief, "Learning DevOps - The complete guide to accelerate collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps", Packt Publishing, 2019.

VI. REFERENCE BOOKS:

1. Joakim Verona, "Practical DevOps", Packt Publishing, 2016.
2. Michael Huttermann, "DevOps for Developers", A press publishers, 2012.
3. Sanjeev Sharma, "The DevOps Adoption Playbook", Published by John Wiley & Sons, Inc.2017.
4. Sanjeev Sharma & Bernie Coyne, "DevOps for Dummies", Published by John Wiley & Sons, Inc.

VII. REFERENCE BOOKS:

1. https://www.geeksforgeeks.org/devops-tutorial/
2. https://www.javatpoint.com/devops
3. https://azure.microsoft.com/en-in/solutions/devops/tutorial
4. https://www.guru99.com/devops-tutorial.html

UNDERTAKING BY STUDENT / PARENT

"To make the students attend the classes regularly from the first day of starting of classes and be aware of the College regulations, the following undertaking form is introduced which should be signed by both student and parent. The same should be submitted to the Dean of Academic".

I, Mr. / Ms. -- joining I Semester / III Semester for the academic year 20 - 20 / 20 - 20 in Institute of Aeronautical Engineering, Hyderabad, do hereby undertake and abide by the following terms, and I will bring the ACKNOWLEDGEMENT duly signed by me and my parent and submit it to the Dean of Academic.

1. I will attend all the classes as per the timetable from the starting day of the semester specified in the institute Academic Calendar. In case, I do not turn up even after two weeks of starting of classes, I shall be ineligible to continue for the current academic year.
2. I will be regular and punctual to all the classes (theory/laboratory/project) and secure attendance of not less than 75% in every course as stipulated by Institute. I am fully aware that an attendance of less than 65% in more than 60% of theory courses in a semester will make me lose one year.
3. I will compulsorily follow the dress code prescribed by the college.
4. I will conduct myself in a highly disciplined and decent manner both inside the classroom and on campus, failing which suitable action may be taken against me as per the rules and regulations of the institute.
5. I will concentrate on my studies without wasting time in the Campus/Hostel/Residence and attend all the tests to secure more than the minimum prescribed Class/Sessional Marks in each course. I will submit the assignments given in time to improve my performance.
6. I will not use Mobile Phone in the institute premises and also, I will not involve in any form of ragging inside or outside the campus. I am fully aware that using mobile phone to the institute premises is not permissible and involving in Ragging is an offence and punishable as per JNTUH/UGC rules and the law.
7. I declare that I shall not indulge in ragging, eve-teasing, smoking, consuming alcohol drug abuse or any other anti-social activity in the college premises, hostel, on educational tours, industrial visits or elsewhere.
8. I will pay tuition fees, examination fees and any other dues within the stipulated time as required by the Institution / authorities, failing which I will not be permitted to attend the classes.
9. I will not cause or involve in any sort of violence or disturbance both within and outside the college campus.
10. If I absent myself continuously for 3 days, my parents will have to meet the HOD concerned / Principal.
11. I hereby acknowledge that I have received a copy of BT23 Academic Rules and Regulations, course catalogue and syllabus copy and hence, I shall abide by all the rules specified in it.

ACKNOWLEDGEMENT

I have carefully gone through the terms of the undertaking mentioned above and I understand that following these are for my/his/her own benefit and improvement. I also understand that if I/he/she fail to comply with these terms, shall be liable for suitable action as per Institute/JNTUH/AICTE/UGC rules and the law. I undertake that I/he/she will strictly follow the above terms.

Signature of Student with Date

Signature of Parent with Date
Name & Address with Phone Number