Hall Ticket No □□□□□□□□□□ Question Paper Code: AIT004

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)

MODEL QUESTION PAPER

B.Tech V Semester End Examinations (Regular), November – 2019

Regulation: IARE–R16

COMPILER DESIGN
(Common to CSE / IT)

Time: 3 Hours **Max Marks:70**

Answer ONE Question from each Unit
All Questions Carry Equal Marks
All parts of the question must be answered in one place only

UNIT – I

1. a) Define regular expression? Write about the identity rules and properties for regular expression. **[7M]**

b) Consider the following Conditional statement: **[7M]**
if (x > 3) then y = 5 else y = 10;
How does lexical analyzer help the above statement in process of compilation?

2. a) Explain in detail about the role of lexical analyzer with the possible error recovery actions? **[7M]**

b) Construct the predictive parsing table for the following grammar **[7M]**
E –> E +T | T
T –> T * F | F
F –> (E) | id
Also show the behavior of the parser on the sentence (i d * i d)+id.

UNIT – II

3. a) Demonstrate stack implementation in shift reduce parsing and explain the terms shift action and reduce action? **[7M]**

b) Find the SLR parsing table for the given grammar: **[7M]**
E->E+E | E*E | (E) | id.
And parse the sentence (a+b)*c.

4. a) Explain the following terms **[7M]**
i) Canonical collection of items
ii) Augmented Grammar
iii) Closure and goto Operation

b) The following grammar for if-then-else statements is proposed to remedy the dangling-else ambiguity: **[7M]**
Stmt → if expr then stmt | matched_stmt
Matched_stmt → if expr then matched_stmt else stmt | other
Show that this grammar is still ambiguous.

UNIT – III

5. a) How would you generate the intermediate code for the flow of control statements?Explain with examples. **[7M]**

b) Generate intermediate code for the following code segment along with the required syntax directed translation scheme: **[7M]**

```
if(a>b)
x=a+b
else
x=a-b
```

Where a and x are of real and b of int type data.

6. a) Write production rules and semantic actions for the following grammar along with annotated parse tree for the string 9-5+4 **[7M]**

expr → expr + term| expr – term | term
term→0|1|2|3|4|5|6|7|8|9

b) Construct a Quadruple, Triple and Indirect Triple for the statement: **[7M]**

a+a*(b-c) +(b-c)*d?

UNIT – IV

7. a) What is the concept of activation record? List (and explain all elements related to activation record. Also differentiate call by copy restore and call by name. **[7M]**

b) Suppose that the type of each identifier is a sub range of integers, for expressions with operators +, -, *, div and mod, as in Pascal. Write type checking rules that assign to each subexpression**.** **[7M]**

8. a) Discuss and analyze about all allocation strategies in run-time storage environment? **[7M]**

b) Define symbol table. Explain different data structures that are used in symbol table organization. **[7M]**

UNIT – V

9. a) Explain the following peephole optimization techniques: **[7M]**

a) Elimination of Redundant Code
b) Elimination of Unreachable Code.

b) Construct the DAG for the following basic block **[7M]**

T1=A+B
T2=C+D
T3=E – T2
T4=T1– T3

Then generate the code for above constructed DAG using only one register.

10. a) Explain in detail about machine dependent code optimization techniques with their drawbacks? **[7M]**

b) Optimize the following code using various optimization techniques: **[7M]**

```
i=1; s=0;
for (i=1; i<=3; i++)
for (j=1;j<=3;j++)
c[i][j]=c[i][j] + a[i][j] + b[i][j].
```

— ◦ ◦ § ◦ ◦ —

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)

COURSE OBJECTIVES:

The course should enable the students to:

I.	Apply the principles in the theory of computation to the various stages in the design of compilers.
II.	Demonstrate the phases of the compilation process and able to describe the purpose and operation of each phase.
III.	Analyze problems related to the stages in the translation process.
IV.	Exercise and reinforce prior programming knowledge with a non-trivial programming project to construct a compiler.

COURSE OUTCOMES:

CO 1	Understand the various phases of compiler and design the lexical analyzer. Demonstrate the phases of the compilation process and able to describe the purpose and operation of each phase.
CO 2	Explore the similarities and differences among various parsing techniques and grammar transformation techniques
CO 3	Analyze and implement syntax directed translations schemes and intermediate code generation.
CO 4	Describe the concepts of type checking and analyze runtime allocation strategies.
CO 5	Demonstrate the algorithms to perform code optimization and code generation.

COURSE LEARNING OUTCOMES:

AIT004.01	Define the phases of a typical compiler, including the front and backend.
AIT004.02	Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.
AIT004.03	Identify tokens of a typical high-level programming language; define regular expressions for tokens and design and implement a lexical analyzer using a typical scanner generator.
AIT004.04	Explain the role of a parser in a compiler and relate the yield of a parse tree to a grammar derivation.
AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; construct a parser for a given context-free grammar.
AIT004.06	Demonstrate Lex tool to create a lexical analyzer and Yacc tool to create a parser.
AIT004.07	Understand syntax directed translation schemes for a given context free grammar.
AIT004.08	Implement the static semantic checking and type checking using syntax directed definition (SDD) and syntax directed translation (SDT).
AIT004.09	Understand the need of intermediate code generation phase in compilers.
AIT004.10	Write intermediate code for statements like assignment, conditional, loops and functions in high level language.
AIT004.11	Explain the role of a semantic analyzer and type checking; create a syntax-directed definition and an annotated parse tree; describe the purpose of a syntax tree.
AIT004.12	Design syntax directed translation schemes for a given context free grammar.
AIT004.13	Explain the role of different types of runtime environments and memory organization for implementation of programming languages.
AIT004.14	Differentiate static vs. dynamic storage allocation and the usage of activation records to manage program modules and their data.
AIT004.15	Understand the role of symbol table data structure in the construction of compiler.

AIT004.16	Learn the code optimization techniques to improve the performance of a program in terms of speed & space.
AIT004.17	Implement the global optimization using data flow analysis such as basic blocks and DAG.
AIT004.18	Understand the code generation techniques to generate target code.
AIT004.19	Design and implement a small compiler using a software engineering approach.
AIT004.20	Apply the optimization techniques to intermediate code and generate machine code

Mapping of Semester End Examinations to Course Learning Outcomes:

SEE Question No		Course Learning Outcomes		Course Outcomes	Blooms Taxonomy Level
1	a	AIT004.01	Define the phases of a typical compiler	CO 1	Understand
	b	AIT004.02	Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.	CO 1	Remember
2	a	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 1	Remember
	b	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 1	Remember
3	a	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 2	Remember
	b	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 2	Remember
4	a	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 2	Remember
	b	AIT004.05	Apply an algorithm for a top-down or a bottom-up parser construction; Construct a parser for a given context-free grammar.	CO 2	Remember
5	a	AIT004.07	Understand syntax directed translation schemes for a context free grammar.	CO 3	Understand
	b	AIT004.11	Explain the role of a semantic analyzer and type checking; create a syntax-directed definition and an annotated parse tree; Describe the purpose of a syntax tree.	CO 3	Understand
6	a	AIT004.09	Understand the need of intermediate code generation phase in compilers.	CO 3	Understand
	b	AIT004.10	Write intermediate code for statements like assignment, conditional, loops and functions in high level language.	CO 3	Understand
7	a	AIT004.11	Explain the role of a semantic analyzer and type checking; create a syntax-directed definition and an annotated parse tree; Describe the purpose of a syntax tree.	CO 4	Understand
	b	AIT004.13	Differentiate static vs. dynamic storage allocation and the usage of activation records to manage program modules and their data.	CO 4	Remember
8	a	AIT004.12	Explain the role of different types of runtime environments and memory organization for implementation of typical programming languages.	CO 4	Understand

	b	AIT004.14	Understand the role of symbol table data structure in the construction of compiler.	CO 4	Understand
9	a	AIT004.15	Learn the code optimization techniques to improve the performance of a program in terms of speed and space.	CO 5	Understand
	b	AIT004.16	Implement the global optimization using data flow analysis such as basic blocks and DAG.	CO 5	Apply
10	a	AIT004.17	Understand the code generation techniques to generate target code.	CO 5	Understand
	b	AIT004.16	Implement the global optimization using data flow analysis such as basic blocks and DAG.	CO 5	Apply

Signature of Course Coordinator **HOD, IT**