COMPILER DESIGN

V Semester: CSE / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSB11	Core	3	1	-	4	30	70	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		

OBJECTIVES:

The students will try to learn:

I. The process of translating a high-level language to machine code required for compiler construction.
II. The Software tools and techniques used in compiler construction such as lexical analyser and parser generators.
III. The data structures used in compiler construction such as abstract syntax trees, symbol tables, three-address code, and stack machines.
IV. The deeper insights into the syntax and semantic aspects of programming languages, dynamic memory allocation and code generation.

COURSE OUTCOMES:

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

1. **Describe the components of a language processing system for the conversion of high level languages to machine level languages.**
2. **Classify the importance of phases of a compiler for constructing a compiler.**
3. **Demonstrate a lexical analyser from a specification of a Language's lexical rules for dividing the programming statements into tokens.**
4. **Construct the derivations , FIRST set , FOLLOW set on the context free grammar for performing the top-down and bottom up parsing methods.**
5. **Distinguish top down and bottom up parsing methods for developing parser with the parse tree representation of the input.**
6. **Construct LEX and YACC tools for developing a scanner and a parser.**
7. **Describe syntax directed definitions & translations for performing Sementic Analysis.**
8. **Classify the different intermediate forms for conversion of syntax translations into Intermediate Code.**
9. **Demonstrate type systems for performing the static and dynamic type checking**
10. **Describe the run-time memory elements for storage allocation strategies which includes procedure calls, local variable allocation, dynamic memory allocation.**
11. **Apply the code optimization techniques on intermediate code form for improving the performance of a program.**
12. **Make use of optimization techniques on basic blocks for reducing utilization of registers in generating the target code.**

MODULE -I INTRODUCTION TO COMPILERS

Introduction to compilers**:** Definition of compiler, interpreter and its differences, the phases of a compiler; Lexical Analysis: Role of lexical analyzer, input buffering, recognition of tokens, finite automata, regular Expressions, from regular expressions to finite automata, pass and phases of translation, bootstrapping, LEX-lexical analyzer generator.

MODULE -II	SYNTAX ANALYSIS
Syntax Analysis: Parsing, role of parser, context free grammar, derivations, parse trees, ambiguity, elimination of left recursion, left factoring, eliminating ambiguity from dangling-else grammar; Types of parsing: Top-down parsing, backtracking, recursive-descent parsing, predictive parsers, LL (1) grammars. Bottom-up parsing: Definition of bottom-up parsing, handles, handle pruning, stack implementation of shift-reduce parsing, conflicts during shift-reduce parsing, LR grammars, LR parsers-simple LR, canonical LR and Look Ahead LR parsers, error recovery in parsing, parsing ambiguous grammars, YACC-automatic parser generator.	
MODULE -III	**SYNTAX-DIRECTED TRANSLATION AND INTERMEDIATE CODE GENERATION**
Syntax-Directed Translation: Syntax directed definitions, construction of syntax trees, S-attributed and L-attributed definitions; Syntax Directed Translation schemes. Intermediate code generation: Intermediate forms of source programs– abstract syntax tree, polish notation and three address code, types of three address statements and its implementation, syntax directed translation into three-address code, translation of simple statements, Boolean expressions and flow-of- Control statements.	
MODULE -IV	**TYPE CHECKING AND RUN TIME ENVIRONMENT**
Type checking: Definition of type checking, type expressions, type systems, static and dynamic checking of types, specification of a simple type checker. Run time environments: Source language issues, Storage organization, storage-allocation strategies, access to nonlocal data on the stack, garbage collection, symbol tables.	
MODULE -V	**CODE OPTIMIZATION AND CODE GENERATOR**
Code optimization: The principle sources of optimization, optimization of basic blocks, loops in flow graphs, peephole optimization. Code Generation: Issues in the Design of a Code Generator, The Target Language, addresses in the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, A Simple Code Generator, register allocation and assignment, DAG representation of basic blocks.	

Text Books:

1. Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, -Compilers-Principles, Techniques and Tools‖, Pearson Education, Low Price Edition, 2004

Reference Books:

1. Kenneth C. Louden, Thomson, ―Compiler Construction– Principles and Practice‖, PWS Publishing 1st Edition ,1997
2. Andrew W. Appel, ―Modern Compiler Implementation C‖, Cambridge University Press, Revised Edition, 2004.
3. Andrew W. Appel, Modern Compiler Implementation C, Cambridge University Press, 2004.

Web References:

1. www.vssut.ac.in/lecture_notes/lecture1422914957.pdf
2. http://csenote.weebly.com/principles-of-compiler-design.html
3. http://www.faadooengineers.com/threads/32857-Compiler-Design-Notes-full-book-pdf-download
4. https://www.vidyarthiplus.com/vp/thread-37033.html#.WF0PhlMrLDc