Kadi Sarva Vishwavidyalaya

Faculty of Engineering & Technology

Second Year Master of Engineering (Computer Engineering)

(Semester-III)

(With effect from: Academic Year 2018-19)

Subject Code: MECE-303-N-D	**Subject Title:** Advanced Complier Design
Pre-requisite	

Teaching Scheme (Credits and Hours)

Teaching Scheme				Total Credit	Evaluation Scheme					
L	T	P	Total		Theory		Mid Sem Exam	CIA	Practical	Total
Hours	Hours	Hours	Hours		Hours	Marks	Marks	Marks	Marks	Marks
04	00	02	06	05	03	70	30	20	30	150

Learning Objectives:

- To understand, the foundation of compiler and mathematical models of computation.
- To understand, what compiler could do and what it couldn’t.
- To check the correctness of the compiler and measure the speed (runtime and compile time)
- To see the degree of optimization.
- To check error reporting and analysis (feedback) to user.
- To check the debugging facility provided by the compiler.

Outline of the Course:

Sr. No	Title of the Unit	Minimum Hours
1	Language Translation Overview	03
2	Lexical Analysis	04
3	Syntax Analysis	06
4	Syntax-Directed Translation	08
5	Memory Allocation , Organization And Memory Management	08
6	Intermediate Code Generation	08
7	Code Generation	08
8	Code Optimization	12
9	Symbol Table Management	07
	Total	**64**

Total hours (Theory): 64
Total hours (Lab): 32
Total hours: 96

Detailed Syllabus:

Sr. No	Topic	Lecture Hours	Weight age (%)
1	**Language Translation Overview** • Overview of language processors, translators, linker, loader. • Types of language processors –assembler, interpreter, compiler. • Overview and use of linker and loader • Compilation phases, back end, front end, pass structureCompiler-Construction Tools	03	04
2	**Lexical Analysis** • The Role of the Lexical Analyser • Regular expressions and regular languages • Input Buffering • Finite automata (RE to NFA, NFA to DFA) Optimization of DFA-Based Pattern Matchers	04	06
3	**Syntax Analysis** • The Role of the Parser • Context-Free Grammars • Top-Down Parsing • Bottom-Up Parsing (Operator-Precedence Parsing, LR Parsers) Using Ambiguous Grammars	06	09
4	**Syntax-Directed Translation** • Syntax-Directed Definitions • Construction of Syntax Trees, • Bottom Up Evaluation of S-Attributed Definitions • L-Attributed Definitions • Top Down Translation • Bottom-Up Evaluation of Inherited Attributes • Recursive Evaluators • Analysis of Syntax-Directed Definitions • Type Systems • Specification of a Simple Type Checker • Equivalence of Type Expressions • Type Conversions • Overloading of Functions and Operators	08	13
5	**Memory Allocation , Organization And Memory Management** • Source Language Issues • Storage Organization • Storage-Allocation Strategies • Access to Non local Names • Parameter Passing and Language Facilities for Dynamic Storage	08	13

	Allocation • Dynamic Storage Allocation Techniques • Activation Tree, Activation Record • Symbol Table • Static, Dynamic And Heap Storage Allocation, • Garbage collection		
6	**Intermediate Code Generation** • Intermediate Languages • Programming statements and intermediate codes: Declarations, Assignment Statements, Boolean Expressions, Switch- Case Statements, Procedure Calls, Loops • Back patching • Types of Intermediate Forms of the Program	08	13
7	**Code Generation** • Issues in the Design of a Code Generator • The Target Machine • RunTime Storage Management • Basic Blocks and Flow Graphs • Next-Use Information • A Simple Code Generator • Register Allocation and Assignment • The DAG Representation of Basic Blocks • Peephole Optimization • Generating Code from DAGs • Dynamic Programming Code-Generation Algorithm • Code Generators	08	13
8	**Code Optimization** • The principal sources of optimization • Common subexpressions, constant propagation, dead code elimination, basic loop optimization, partial redundancy elimination, SSA (static single assignment), induction variables and reduction in strength • Register allocation and assignment • Data flow analysis: The Data-Flow Abstraction, The Data-Flow Analysis Schema, Data-Flow Schemas on Basic Blocks, Reaching Definitions, Live-Variable Arlalysis, Available Expressions, Iterative data flow analysis, lattices of flow function, control-tree based data flow analysis • Control flow analysis: Approaches to control flow analysis, Depth first search, Breadth first search, Preorder traversal, Postorder traversal, Loops in flow graphs, Reducibility, Interval analysis and control trees	12	18

9	**Symbol Table Management** • General concepts • Symbol Table as a data structure • Various operations performed on Symbol Table • Symbol table organizations for blocked structured language and non-blocked structured language	07	11
	Total	64	100

Instructional Method and Pedagogy:

- At the start of course, the course delivery pattern, prerequisite of the subject will be discussed.
- Lectures will be conducted with the aid of multi-media projector, black board, OHP etc.
- Attendance is compulsory in lecture and laboratory which carries 10 marks in overall evaluation.
- One internal exam will be conducted as a part of internal theory evaluation.
- Assignments based on the course content will be given to the students for each unit and will be evaluated at regular interval evaluation.
- Surprise tests/Quizzes/Seminar/tutorial will be conducted having a share of five marks in the overall internal evaluation.
- The course includes a laboratory, where students have an opportunity to build an appreciation for the concepts being taught in lectures.
- Experiments shall be performed in the laboratory related to course contents.

Learning Outcome:

On successful completion of this course, the student should be able to:

- be able to understand the principals of compiler design and will be able to generate the basic compiler
- become familiar with front and back end of the compiler
- be able to use Lex and YACC tool.
- be able to perform lexical analysis and various parsing technique.

Reference Books:

1. Compilers: principles, techniques & tools by Alfred V Aho, Monica S. lam, Ravi Sethi, Jeffrey D. Ullman, Pearson
2. Advanced compiler design & implementation by Steven S. Muchnick, Morgan Kaufmann
3. Optimizing Compilers for Modern Architectures by Randy Allen & Ken Kennedy, Morgan Kaufmann
4. High Performance Compilers for Parallel Computing by Michael Wolfe, Addison-Wesley
5. Compiler Writing by Tremblay and Sorenson, BS Publicatio

List of experiments:

Sr. No.	Name of Experiment
1	Implement a C program to identify keywords, identifiers and numbers using finite automata.
2	Write a lex program to identify numbers, words and other characters and generate tokens for each.
3	Write a lex program to count the number of characters, words, lines, new lines, tabs and whitespaces in the given input
4	Write a lex program that will replace the word “Hello” with “ldrp” if the line starts with the letter ‘a’ and with “college” if it starts with ‘b’.
5	Write a lex program to display the comments from given input file. Provide the input file as command line argument.
6	Generate a lexer for C program.
7	Write a C program to eliminate left recursion and perform left factoring from given productions.
8	Write a C program to implement any one top-down parser.
9	Write a C program to implement any one bottom-up parser.
10	Implementation of Yacc programs. a. Write a Yacc program for desktop calculator with ambiguous grammar. b. Write a Yacc program for desktop calculator with ambiguous grammar and additional information
11	Write a Yacc program for calculator with unambiguous grammar.
12	Write a program to generate three address code of any one loop.