Subject Name: Compiler Design

Subject Code: CE701-N

Teaching Scheme (Credits and Hours)

Teaching scheme				Total Credit	Evaluation Scheme					
L	T	P	Total		Theory		Mid Sem Exam	CIA	Pract.	Total
Hrs	Hrs	Hrs	Hrs		Hrs	Marks	Marks	Marks	Marks	Marks
04	00	02	06	5	3	70	30	20	30	150

Learning Objectives:

The objective of this course is to introduce students to the following concepts underlying the design and implementation of compilers.

- Describe the steps and algorithms used by compilers.
- Recognize the underlying formal models such as finite state automata, push-down automata and their connection to language definition through regular expressions and grammars.
- Discuss the effectiveness of optimization.
- Explain the impact of a separate compilation facility and the existence of program libraries on the compilation process.

Outline of the Course:

Sr. No	Title of the Unit	Minimum Hours
1	Introduction to Compiling	4
2	Lexical Analyzer	6
3	Parsing Theory • Syntax Analyzer • Syntax Directed Translation	14
4	Error Recovery	4
5	Type Checking	4
6	Run Time Environments	5
7	Intermediate Code Generation	5
8	Code Generation	5
9	Code Optimization	7
10	Introduction to Language processors and system software	10

Total hours (Theory): 64

Total hours (Practical): 32

Total hours: 96

Detailed Syllabus:

Sr. No	Topic	Lecture Hours	Weightage(%)
1	**Introduction to Compiling** • Overview of the Translation Process- A Simple Compiler, Difference between interpreter, assembler and compiler • Overview and use of linker and loader , • types of Compiler, • Analysis of the Source Program, • The Phases of a Compiler, • Cousins of the Compiler, The Grouping of Phases, • Front-end and Back-end of compiler, • pass structure • A simple one-pass compiler: overview	04	06
2	**Lexical Analyzer** • Introduction to Lexical Analyzer, • Input Buffering, • Specification of Tokens, • Recognition of Tokens, • A Language for Specifying Lexical Analyzers, • Finite Automata From a Regular Expression, • Design of a Lexical Analyzer Generator, • Optimization of DFA	06	09
3	**Parsing Theory- Syntax Analyzer** • The role of a parser • Context free grammars • Top Down and Bottom up Parsing Algorithms, • Top-Down Parsing, • Bottom-Up Parsing, • Operator-Precedence Parsing, • LR Parsers, • Using Ambiguous Grammars, • Parser Generators, • Automatic Generation of Parsers.	10	16
	Parsing Theory- Syntax Directed Translation •Syntax-Directed Definitions, •Construction of Syntax Trees, •Bottom-Up Evaluation of S-Attributed Definitions, •L-Attributed Definitions, •Syntax directed definitions and translation schemes	04	06

4	**Error Recovery** • Error Detection & Recovery, • Ad-Hoc and Systematic Methods	04	06
5	**Type Checking** • Type systems • Specification of a simple type checker • Type conversions	04	06
6	**Run Time Environments** • Source Language Issues, • Storage Organization, • Storage-Allocation Strategies, • Parameter Passing, • Symbol Tables, • Language Facilities for Dynamic Storage Allocation, • Dynamic Storage Allocation Techniques.	05	08
7	**Intermediate Code Generation** • Different Intermediate Forms, • Implementation of Three Address Code • Intermediate code for all constructs of programming languages (expressions, if-else, loops, switch case etc.)	05	08
8	**Code Generation** • Issues in the Design of a Code Generator • Basic Blocks and Flow Graphs • A Simple Code Generator • Register Allocation and Assignment • The DAG Representation of Basic Blocks • Peephole Optimization • Dynamic Programming Code-Generation Algorithm	05	08
9	**Code Optimization** • Global Data Flow Analysis, • A Few Selected Optimizations like Command Sub Expression Removal, Loop Invariant Code Motion, Strength Reduction Etc. • Optimization of basic blocks	07	11
10	**Introduction to Language processors and system software** • Macros and Macro Processors: Macro Definition and Call , Macro Expansion • Assemblers: Elements of Assembly Language Programming, A Simple Assembly Scheme ,Pass Structure of Assemblers, Design of a Two Pass Assembler • Software Tools for Program Development • Editors • System software: linker/loader	10	16
	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.

STUDENTS LEARNING OUTCOMES:

On successful completion of the course, the student will:

- Understand how the design of a compiler requires most of the knowledge acquired during their study.
- Develop a firm and enlightened grasp of concepts learned earlier in their study like higher level programming, assemblers, automata theory, and formal languages.
- Apply the ideas, the techniques, and the knowledge acquired for the purpose of other language processor design.
- Working skills in theory and application of finite state machines, recursive descent, production rules, parsing, and language semantics.
- Know about the powerful compiler generation tools, which are useful to the other non-compiler applications
- Be able to compare various system software related to the given system
- Be able to understand the concepts required to develop the system software

Reference Books:

1. Compilers, Principles, Techniques and Tools by A.V. Aho, R. Sethi and J.D.Ullman, Pearson
2. D. M. Dhamdhere, "Systems Programming and Operating Systems", Second Revised Edition, Tata McGraw-Hill, 1999.
3. Advanced compiler Design Implementation by Steven S. Muchnick
4. The Compiler Design handbook: Optimization and Machine Code Generation by Y. N. Shrikant and Priti Shankar, Second Edition
5. Charles N. Fischer, Richard J. leBlanc, Jr.- Crafting a Compiler with C, Pearson Education, 2008.

List of Practical:

Sr.NO	Practical
1	Implement a C program to identify keywords and identifiers using finite automata.
2.	Implement a C program to identify whether the production is left recursive or not and eliminate left recursion if it is applicable.
3.	Implement a C program to remove left factoring.
4.	**Implementation of lex programs:**
	Write a lex program to identify numbers, words and other characters and generate tokens for each.
	Write a lex program to identify all occurrences of "LDRP" and replace it with "COLLEGE".
	Write a lex program to display the length of each word.
	Write a lex program to convert the lowercase first letter of the string to upper case and upercase first letter of the string to lowercase.
	Write a lex program to count the number of characters, words and lines in the given input.
	Write a lex program to add line numbers to every line of a input file.
	Write a lex program that read the numbers and add 3 to the numbers if the number is divisible by 7.
	Write a lex program to remove empty lines.
	Write a lex program to identify words followed by punctuation marks.
	Write a lex program to display the comments from given input file. \\|*
	Write a lex program to identify all the lexemes from input file that follow the given RE. Provide the RE and input file as command line arguments.
	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'.
	Generate a lexer for C program.
5	**Implementation of Yacc programs.**
	Write a Yacc program for desktop calculator with ambiguous grammar.
	Write a Yacc program for desktop calculator with ambiguous grammar and additional information.
	Write a Yacc program for calculator with unambiguous grammar.
6	Implement pass-I of a two pass assembler