Kadi Sarva Vishwavidyalaya

Faculty of Engineering & Technology

Third Year Bachelor of Engineering (Information Technology)

(In Effect From Academic Year 2019-20)

Subject Code: IT605E-N	**Subject Title:** Compiler Design
Pre-requisite	Basic knowledge of formal languages and automata theory

Teaching Scheme (Credits and Hours)

Teaching scheme				Total Credit	Evaluation Scheme					Total
L	T	P	Total		Theory		Mid Sem Exam	CIA	Pract.	Total
Hrs	Hrs	Hrs	Hrs		Hrs	Marks	Marks	Marks	Marks	Marks
03	00	02	05	04	03	70	30	20	30	150

Course objective:

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

- Description of the phases and algorithms used by compilers.
- Recognition of the underlying formal models such as finite state automata, push-down automata and their application in design of the compiler 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.
- Make the students be able to thoroughly understand the concepts and implementation required to design compilers.

Outline of the Course:

Sr. No	Title of the Unit	Minimum Hours
1	Introduction to Compiler	5
2	Lexical Analysis	7
3	Parsing Theory • Syntax Analysis • Syntax Directed Translation	15
4	Type Checking	3
5	Run Time Environments	4
6	Intermediate Code Generation	4
7	Code Optimization	5
8	Code Generation	5

Total hours (Theory):48
Total hours(Lab):32
Total hours:80

Detailed Syllabus:

Sr. No	Topic	Lecture Hours	Weight age(%)
1	**Introduction to Compiler** • 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	5	10
2	**Lexical Analysis** • 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 • Error recovery in Lexical Analyzer	7	15
3	**Parsing Theory** **Syntax Analysis:** • The role of the parser • Context free grammars • Top-Down Parsing (Recursive Descent Parser, Predictive Parser) • Bottom-Up Parsing (Operator-Precedence Parsing,LR Parsers) • Error Recovery in Parsers • Using Ambiguous Grammars • Parser Generators **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	15	30
4	**Type Checking** • Type systems • Specification of a simple type checker • Type conversions	3	5

5	**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	4	10
6	**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.)	4	10
7	**Code Optimization** • The Principal Sources of Optimization • Optimization of Basic Blocks • Loops in Flow Graphs • Global Data Flow Analysis • Optimization Techniques like Command Sub Expression Elimination, Loop Invariant Code Motion, Strength Reduction, Induction Variables etc. • Code-Improving Transformations	5	10
8	**Code Generation** • Issues in the Design of a Code Generator • The Target Machine • Basic Blocks and Flow Graphs • 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	5	10
	Total	48	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 the course, the student will be able to:

- understand how the compilersare constructed for different programming languages.
- apply the ideas, the techniques and the knowledge acquired for the purpose of other language processor design.
- explore the applications of finite state machines, production rules, parsing and language semantics.
- study the powerful compiler generation tools like Lex and Yacc.

e-Resources:

- https://nptel.ac.in/courses/106108113/

Reference Books:

1. A.V. Aho, R. Sethi and J.D.Ullman, "Compilers, Principles, Techniques and Tools",Pearson
2. J.P. Bennet, "Introduction to Compiler Techniques", Second Edition, Tata McGraw Hill
3. Charles N. Fischer, Ron K. Cytron, Richard J. LeBlanc, Jr., "Crafting a Compiler", PearsonEducation
4. Kenneth C. Louden, "Compiler Construction: Principles and Practice", Thompson Learning
5. John R. Levine, Tony Mason, Doug Brown, "lex & yacc", O'Reilly

List of experiments

No	Name of Experiment
1	Implement a C program to identify keywords, numbers and identifiers using finite automata and without using finite automata.
2	a. Write a lex program to identify and generate tokens for numbers (int and float both), words and other characters and display the length of each. b. Write a lex program to identify words followed by punctuation marks (take input file using command line arguments).
3	a. Write a lex program to change the case of the first letter of every word. b. Write a lex program to count number of consonants and vowels from given input string.
4	a. Write a lex program to count the number of characters, words, lines and empty lines in the given input. Also it should remove empty lines. b. Write a lex program to display the comments from given input file.
5	a. Write a lex program to identify all occurrences of "LDRP" and replace it with "COLLEGE". b. Write a lex program that will replace the word "Hello" with "ldrp" if the line begins with the letter 'a' and with "college" if it begins with 'b' (take input file using command line arguments).
6	a. Write a lex program that copies an input file while adding 3 to every positive number divisible by 7. (The program should work properly with negative numbers. It should not alternate floating point numbers like 49.63 or variable names like a7). b. Write a lex program which histograms the lengths of words, where a word is defined as a string of letters.

7	Generate a lexer for C program.
8	Write a C program to eliminate left recursion from a production.
9	Write a C program to apply left factoring to a production.
10	Write a C program to implement recursive descent parser.
11	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 andadditional information about precedence and associativity of operators.
12	Write a Yacc program for desktop calculator withunambiguous grammar.