SARVAJANIK UNIVERSITY
Sarvajanik College of Engineering and Technology
Bachelor of Technology

Year: B. Tech IV (Semester VII)

Subject Name: Automata Theory and Compiler Design **Subject Code:** BTIT13701
Type of course: Professional Core Course
Prerequisite (if any): - Data Structures, A basic course of Programming

Rationale: To design a compiler, is inevitable to grasp the knowledge of various types of grammar, lexical analysis, YACC, FSM(Finite State Machines), and correlative concepts of languages. This subject focuses on the formal relationships among machines, languages, and grammars and computational problems. It then provides an in-depth view of translation, optimization, and compilation of the entire source program. It also focuses on various designs of compilers and the structuring of various phases of compilers.

Teaching and Examination Scheme:

Teaching Scheme				Theory Marks			Practical Marks		Total
L	T	P	C	TEE	CA1	CA2	TEP	CA3	
3	0	2	4	60	25	15	30	20	150

CA1: Continuous Assessment (assignments / projects / open book tests / closed book tests) CA2: Sincerity in attending classes / class tests / timely submissions of assignments / self-learning attitude / solving advanced problems TEE: Term End Examination TEP: Term End Practical Exam (Performance and viva on practical skills learned in course) CA3: Regular submission of Lab work / Quality of work submitted / Active participation in lab sessions / viva on practical skills learned in course.

Contents:

Sr. No.	Contents	Total Hours
1.	**Introduction to Automata** **Finite Automata:** Formal Language and Regular Expressions, Finite Automata, Deterministic Finite Automata(DFA), Non-Deterministic Finite Automata(NFA), Construction NFA from regular expression, Conversion of NFA to DFA, Minimizing the Number of States in a Finite Automaton. **Context-Free Grammars:** Context-free grammars and languages, Regular Languages and Regular Grammars, Derivation Trees and Ambiguity, Normal Forms for Context-Free Grammars, Chomsky Normal Form, Greibach Normal Form, Derivation Trees, Leftmost and Rightmost Derivation of Strings and Sentential Forms, Ambiguity, Left recursion and Left factoring. **Pushdown Automata and Turing Machines:** Pushdown Automata(PDA)-Definitions and Examples, Deterministic Pushdown Automata, PDA from CFG, CFG from PDA, A General Model of Computation,	15

	Turing Machines as Language Acceptors	
2.	**Overview of the Compiler:** Overview of Compilers, Phases of a Compiler, Symbol-Table Management, The Grouping of phases into Passes, Compiler Construction Tools, A simple Syntax-directed Translator, A model of a Compiler Front End, Postfix Notation, Synthesized Attributes, Parsing, Designing a Predictive Parser, A Translator for Simple Expressions, A Lexical Analyzer, Removal of White spaces and comments, Intermediate Code Generation	06
3.	**Lexical Analysis:** The Role of the Lexical Analyzer, Specification of Tokens, Recognition of Tokens, Input Buffering, Elementary Scanner Design and its Implementation, The Lexical-Analyzer Generator Lex, Applying concepts of Finite Automata for recognition of tokens	04
4.	**Syntax Analysis:** The Role of the Parser, Representative Grammars, Syntax Error Handling, Error-Recovery Strategies, Top Down Parsing, FIRST And FOLLOW, LL(1) Grammars, Predictive Parsing Algorithms, Bottom-Up Parsing, Reductions, Handle Pruning, Shift-Reduce Parsing, LR Parsers, Constructing Simple LR Parsing Tables, Viable Prefixes, Canonical LR(1) And LALR Parsing Tables, Operator-Precedence Parsing, The Parser Generator YACC	12
5.	**Intermediate-Code Generation and Code Optimization:** Variants of Syntax Trees, Three-Address Codes- Quadruples, Triples, Types and Declarations, Translation of Expressions, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, Directed Acyclic Graph(DAG) representation of basic blocks, Dead Code Elimination, Peephole Optimization, Machine-Independent Optimizations: Global Common Subexpressions, Copy Propagation, Dead-Code Elimination, Code Motion, Induction variable and Reduction in strength	08

Suggested Specification table with Marks (Theory): (For B. Tech only)

Distribution of Theory Marks					
R Level	**U Level**	**A Level**	**N Level**	**E Level**	**C Level**
10	15	30	5	-	-

Legends: R: Remembrance; U: Understanding; A: Application, N: Analyze and E: Evaluate C: Create (Revised Bloom's Taxonomy)

Reference Books:

Sr no	Title of book /article	Author(s)	Publisher and details like ISBN	Year of publication	Publication Edition
1	Introduction to Languages and the Theory of Computation	John Martin,	Tata McGraw Hill	2003	3rd Edition
2	An Introduction To Automata Theory And Formal Languages	Adesh K. Pandey	S.K. Kataria& Sons	2010	5th Edition
3	Compiler Tools Techniques	Alfred V.Aho, Ravi Sethi, J. D. Ullman	Addison Wesley	2005	1st Edition
4	Modern Compiler Design	Dick Grune, Henri E. Bal, J.H.Jacobs, Koen G Langendoen	Springer	2012	2nd Edition

Course Outcomes (CO):

Sr. No.	CO statements	Marks % weightage
CO-1	Read and write finite automata and grammars as well as able to build push-down automata and Turing machine for programming language constructs.	30%
CO-2	Understand the basic concept of compiler design and its different phases which will be helpful to construct new tools like LEX, YACC, etc.	25%
CO-3	Identify and select suitable parsing strategies for a compiler for various cases. Knowledge in alternative methods (top-down or bottom-up, etc.)	30%
CO-4	Understand the backend of the compiler: intermediate code, Code optimization Techniques, and Apply the code optimization techniques to improve the space and time complexity of programs while programming.	15%

List of Practicals:

1. Develop a 'C' Program finds white spaces, number of newline characters from the given input.
2. Implement a lexical analyzer (Scanner program) to recognize identifiers, keywords and constants from the given input file and store them separately.
3. Write a C program to simulate lexical analyzer to validate arithmetic operators, relational operators and logical operators.

4. Convert the following regular expression (R.E.) into DFA and Write a‘C’ program to simulate the DFA for given input Strings.
   (i) a(a|b)* ab. (ii) digit(digit)*(.digit(digit)*|ε)
5. Implement Recursive Descent Parser program in ‘C’ for the following Grammar.

   P ---> E '#'
   E ---> T {('+'|'-') T}
   T ---> S {('*'|'/') S}
   S ---> F '^' S | F
   F ---> D | '(' E ')'
   D ---> 0|1|……|9.
   Write a program in a way that it will trace the processing of different non-terminals of above grammar for given input string.
6. Write a program to remove left recursion from a given grammar.
7. Write a program to left factor the given grammar.
8. Finding “First” set
   Input: The string consists of grammar symbols.
   Output: The First set for a given string.
   Explanation:
   The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the First set of the given string.
9. Write a program to illustrate steps of LL (1) parser for the given parsing table.
10. Implement a C program to implement operator precedence parsing.
11. Generate 3-tuple intermediate code for given infix expression.
12. Study the following compiler construction tools:
    a) LEX b) YACC