GUJARAT TECHNOLOGICAL UNIVERSITY

Program Name: Bachelor of Engineering
Level: UG
Subject Code : 3174801
Subject Name : Compiler Design

w. e. f. Academic Year:	A.Y. 2025-26
Semester:	VII
Category of the Course:	PCC

Prerequisite:	Algorithms, Data Structures, Assembly Language Program, Theory of Computation, C/C++ Programming Skills
Rationale:	Compiler Design is a fundamental subject of Computer Engineering. Compiler design principles provide an in-depth view of translation, optimization and compilation of the entire source program. It also focuses on various designs of compiler and structuring of various phases of compiler. It is inevitable to grasp the knowledge of various types of grammar, lexical analysis, yacc, FSM(Finite State Machines) and correlative concepts of languages.

Course Outcome:
After Completion of the Course, Student will able to:

No	Course Outcomes	RBT Level
1	Understand fundamental principles of compiler design, including formal languages and automata theory.	U
2	Apply lexical and syntax analysis techniques to build efficient front-end phases of compilers.	Ap
3	Analyze and implement parsing strategies such as LL, LR, and operator precedence parsers.	An
4	Demonstrate knowledge of intermediate code generation, optimization techniques, and error handling methods.	U
5	Evaluate runtime environments and apply instruction-level parallelism techniques in code generation..	E

**Revised Bloom's Taxonomy (RBT)*

GUJARAT TECHNOLOGICAL UNIVERSITY

Program Name: Bachelor of Engineering
Level: UG
Subject Code : 3174801
Subject Name : Compiler Design

Teaching and Examination Scheme:

Teaching Scheme (in Hours)			Total Credits L+T+ (PR/2)	Assessment Pattern and Marks				Total Marks
				Theory		Tutorial / Practical		
L	T	PR	C	ESE (E)	PA / CA (M)	PA/CA (I)	ESE (V)	
4	0	2	5	70	30	20	30	150

Course Content:

Unit No.	Content	No. of Hours
1.	**Overview of the Compiler and its Structure** Introduction to language processors and their real-world applications. Detailed examination of compiler structure, internal working, and the scientific principles behind compiler construction. Fundamental understanding of interpreters and assemblers, including distinctions between compilers and interpreters. Compilation workflow from source code to target code. Overview of related tools like assemblers and linkers ("cousins" of compilers) and various types of compilers.	05
2.	**Lexical Analysis**: Exploration of the lexical analyzer's role in the compilation process. Token specification and recognition techniques. Understanding input buffering mechanisms. Introduction to basic scanner design, including practical implementation using Lex. Application of finite automata concepts in token identification and lexical pattern recognition.k.	05
3	**Syntax Analysis:**Comprehensive understanding of parsers and context-free grammars (CFGs). Techniques to handle left recursion and apply left factoring. Study of top-down and bottom-up parsing approaches, including operator precedence parsing and LR parsing techniques. Managing ambiguous grammars and using parser generators for automatic parser creation. Syntax-directed definitions and the construction of syntax trees. In-depth focus on S-attributed and L-attributed definitions and the use of translation schemes in syntax-directed translation.	12
4	**Error Recovery:** Study of various techniques for detecting and recovering from errors during compilation. Comparison between ad-hoc methods and systematic approaches to maintain parser robustness and ensure reliable error handling.	05

5.	**Intermediate-Code Generation** :Detailed exploration of intermediate representations including different forms of syntax trees and three-address code. Handling data types and declarations, translating expressions to intermediate code, and performing type checking. Application of syntax-directed translation techniques and the use of attributed grammars and definitions for code generation.	06
6.	**Run-Time Environments** Analysis of runtime behavior of programs, covering challenges posed by source language features. Understanding storage organization, stack-based memory allocation, access methods for non-local variables, and efficient heap memory management during program execution.	06
7.	**Code Generation and Optimization :**Discussion on critical design issues in code generation including selecting an appropriate target language and managing memory addresses. Construction of basic blocks and control flow graphs, with emphasis on optimizing basic blocks. Implementation of a simple code generator and examination of both machine-dependent and machine-independent optimization strategies. Techniques for error detection and recovery during code generation.	06
8.	**Instruction-Level Parallelism:**Introduction to modern processor architectures and how instruction-level parallelism is leveraged. Understanding constraints in code scheduling and strategies for basic-block scheduling. Overview of assembler pass structures and their role in optimizing code execution.	06

Suggested Specification Table with Marks (Theory):

Distribution of Theory Marks (in %)					
R Level	**U Level**	**A Level**	**N Level**	**E Level**	**C Level**
–	**30**	**25**	**20**	**15**	**10**

Where U: Understanding; A: Application, N: Analyze and E: Evaluate C: Create (as per Revised Bloom's Taxonomy)

References/Suggested Learning Resources:

(a) Books:

1. Compiler Tools Techniques - A.V.Aho, Ravi Sethi, J.D.Ullman, Addison Wesley
2. The Theory And Practice Of Compiler Writing - Trembley J.P. And Sorenson P.G. Mcgraw-Hill

Reference Books:

1. Modern Compiler Design - Dick Grune, Henri E. Bal, Jacob, Langendoen, WILEY India
2. Compiler Construction - Waite W.N. And Goos G., Springer Verlag
3. Compiler Construction-Principles And Practices - D.M.Dhamdhere, Mcmillian
4. Principles of Compiler Design, V. Raghavan, McGrawHill

Sample List of Experiments:

SrNo	Title of Experiment
1	Implementation of Finite Automata and String Validation
2	Introduction to LexTool.
3	Implement following Programs UsingLex a. Generate Histogram of words b. Caesar Cypher c. Extract Single And Multi Line Comments From C Program
4	Implement Following Programs Using Lex a. Convert Roman To Decimal b. Check Weather Given Statement Is Compound Or simple c. Extract Html Tags From.html file
5	Implementation of Recursive Descent Parser Without Backtracking Input: The string to be parsed. Output: Whether String Parsed Successfully Or Not.Explanation: Students have to implement the recursive procedure for DP for atypical grammar.The production no. are displayed as they are used to derive the string.
6	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.

7	Generate3-tuple intermediate code forgiven infix expression
8	Extract Predecessor and Successor from given Control Flow Graph
9	Introduction toYACC and generate Calculator Program
10	Finding“Follow”set Input: The String Consists Of Grammar Symbols. Output: The Follow 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 Follow set of the given string.
11	ImplementaCprogram for constructing LL(1)parsing.
12	Implementa Cprogram to implement LALRparsing.
13	Implementa Cprogram to implement to perator precedence parsing.

****