NIRMA UNIVERSITY

Institute:	**Institute of Technology, School of Technology**
Name of Programme:	**BTech CSE**
Course Code:	4CS501CC25
Course Title:	**Principles of Compiler Design**
Course Type:	**Core**
Year of Introduction:	**2025-26**

L	T	Practical Component				C
		LPW	PW	W	S	
3	0	2	-	-	-	4

Course Learning Outcomes (CLO):

At the end of the course, the students will be able to:

1. summarise the functionalities of various phases of the compiler (BL2)
2. apply language theory concepts to various phases of compiler design (BL3)
3. select the appropriate optimization technique for the compilation process (BL5)
4. implement various compiler phases using the appropriate compiler design tools. (BL6)

Unit	Contents	Teaching Hours (Total 45)
Unit-I	**Introduction:** Overview of principles and significance of compiler design, Structure of compiler, Types of compilers and their applications, The role of language theory in compiler design	03
Unit-II	**Formal Languages and Automata**: Introduction to formal languages and their types, the need for automata and formal languages in compiler design, Deterministic and Non-Deterministic Finite Automata, Conversion from NFA to DFA, Finite Automata, Regular Expression to Automata	06
Unit-III	**Lexical Analysis:** The role of a Lexical Analyzer, Input Buffering, Specifications of Tokens, Recognition of tokens, Lexical Analyzer Generator	05
Unit-IV	**Syntax Analysis:** Context-free Grammar, Top-down Parsing, Bottom-up Parsing, LR Parsers, Error Recovery, parsing for ambiguous grammar, Parsing Generator Tools	10
Unit-V	**Semantic Analysis and Intermediate Code Generation:** Typical Semantic errors, Static and Dynamic Checks, Syntax directed definitions (SDD) & Translation schemes (SDT), Type checking, Syntax Directed Translation Schemes, Type checking, Type Checking, Intermediate code representations, Three Address Codes, Control Flow, Back patching	13
Unit-VI	**Runtime Environment:** Storage Organization, Stack Allocation and Heap Management	02

Unit-VII	**Code Generation and Optimization:** Issues in code generation, Data Flow and Control Flow, Peephole Optimization, Register Allocation, Machine independent optimization techniques, Introduction to cross-compiler	06

Self-Study:

The self-study contents will be declared at the commencement of the semester. Around 10% of the questions will be asked from self-study contents

Suggested Readings/ References:

1. Aho, Lam, Ullman, Sethi, Compilers, *Principles, Techniques and Tools,* Pearson
2. Jean-Paul Trembly & Paul G Sorenson, *The Theory and Practice of Compiler writing*, McGraw Hill
3. Keith D Cooper & Linda Torczon, *Engineering a Compiler*, Elsevier
4. John C. Martin, *Introduction to Languages and Theory of Computation*, McGraw Hill
5. Michael Sisper, *Theory of Computation*, Thomson

Suggested List of Experiments:

Sr.	Name of Experiments/Exercises	Hours
1	Getting acquainted with Lexical Analyzer generator tool lex/flex to recognize tokens from the given code fragment	02
2	**To implement Lexical Analyzer:** Define source programming language and its constructs (one control construct, two arithmetic operators, one loop construct). Use the lex/flex tool to generate a token stream for a given character stream after eliminating comments and other extraneous tokens. Report lexical errors with line numbers.	02
3	**To implement symbol table generation:** Extend experiment 2 to construct a symbol table construct (name, data type, address), and add all unique identifier names to the symbol table.	02
4	**To implement Syntax Analyzer:** Write Context Free Grammar for your defined programming constructs. Use Lex and YACC tools to validate the grammar of the input program.	04
5	**To implement Error recovery in syntax analyzer:** Apply error recovery in experiment 4 to list syntax errors in the input program.	04
6	**To Implement a semantic analyzer:** Write semantics for declaration statements to update the data type of symbols in the symbol table. Also, perform a semantic check to verify and report the use of undeclared variables and redeclaration of variables	04
7	**To implement three address code generators for control statements:** Write semantic rules to generate three address codes for control statements	04
8	To implement three address code generator part II: write semantic rules to generate	04
9	**To implement Assembly code generator.:** Implement getreg() to allocate registers to variables in given three address code.	02
10	**To implement Code Optimization techniques:** Implement any code optimization technique.	02