COURSE CONTENT

COMPILER DESIGN								
VI Semester: CSE / CSD / CSM								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
2460517	Core	3	1	0	4	40	60	100
Contact Classes: 45	Tutorial Classes: 15	Practical Classes: Nil				Total Classes: 60		
Prerequisites: Formal Languages and Automata Theory								

Course Overview:

The Compiler Design course helps students understand how a programming language is converted into a form that a computer can actually execute. It explains each stage of a compiler, starting from reading the source code to producing optimized machine code. Students learn how programs are analyzed, checked for errors, and translated efficiently. The course also introduces important concepts like syntax, memory management, and code optimization. Overall, it builds a strong foundation for understanding how programming languages and system software work internally.

Course Objectives:

1. To understand the structure, phases, and role of a compiler in translating programming languages.
2. To learn lexical analysis techniques using regular expressions, finite automata, and lexical analyzer generators.
3. To study syntax analysis methods, grammar construction, and parsing techniques for language processing.
4. To explore syntax-directed translation, intermediate code generation, type checking, and run-time environments.
5. To analyze and apply code generation strategies and machine-independent optimization techniques for efficient execution.

Course Outcomes: After Completion of the Course, Students should be able to

1. Implement lexical analyzers using regular expressions, finite automata, and Lex tools.
2. Apply parsing techniques (top-down, bottom-up, and LR) for analyzing context-free grammars.
3. Construct syntax-directed translations and generate intermediate code for programming constructs.
4. Design run-time environments and apply code generation strategies with optimization.
5. Perform machine-independent code optimization using data-flow analysis and loop optimizations.

UNIT - I: Introduction: -

The structure of a compiler, the science of building a compiler, programming language basics Lexical Analysis: The Role of the Lexical Analyzer, Input Buffering, Recognition of Tokens, The Lexical-Analyzer Generator Lex, Finite Automata, From Regular Expressions to Automata, Design of a Lexical-Analyzer Generator, Optimization of DFA-Based Pattern Matchers. [10]

UNIT - II: Syntax Analysis: -

Introduction, Context-Free Grammars, writing a Grammar, Top-Down Parsing, Bottom-Up Parsing, Introduction to LR Parsing: Simple LR, More Powerful LR Parsers, Using Ambiguous Grammars and Parser Generators. [10]

UNIT - III: Syntax-Directed Translation: -

Syntax-Directed Definitions, Evaluation Orders for SDD's, Applications of Syntax-Directed Translation, Syntax-Directed Translation Schemes, Implementing L-Attributed SDD's. Intermediate-Code Generation: Variants of Syntax Trees, Three-Address Code, Types and Declarations, Type Checking, Control Flow, Switch-Statements, Intermediate Code for Procedures. [10]

UNIT - IV: Run-Time Environments: -

Stack Allocation of Space, Access to Nonlocal Data on the Stack, Heap Management, Introduction to Garbage Collection, Introduction to Trace-Based Collection. Code Generation: Issues in the Design of a Code Generator, The Target Language, Addresses in the Target Code, Basic Blocks and Flow Graphs, Optimization of Basic Blocks, A Simple Code Generator, Peephole Optimization, Register Allocation and Assignment, Dynamic Programming. [10]

UNIT - V: Machine-Independent Optimization: -

The Principal Sources of Optimization, Introduction to Data-Flow Analysis, Foundations of Data-Flow Analysis, Constant Propagation, Partial- Redundancy Elimination, Loops in Flow Graphs. [8]

TEXT BOOKS:

1. Compilers: Principles, Techniques and Tools, Second Edition, Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffry D. Ullman.

REFERENCE BOOKS:

1. Lex&Yacc – John R. Levine, Tony Mason, Doug Brown, O'reilly
2. Compiler Construction by Louden, Thomson

ELECTRONIC RESOURCES:

1. https://www.geeksforgeeks.org/compiler-design-tutorials/
2. https://www.tutorialspoint.com/compiler_design/index.htm
3. https://www.javatpoint.com/compiler-design
4. https://www.programiz.com/compiler-design
5. https://www.coursera.org/learn/compilers

MATERIALS ONLINE:

1. Course template
2. Tutorial question bank
3. Tech talk and Concept Video topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. E-Learning Readiness Videos