Theory of Computation and Compiler Design (84 hours)

Course Code	Course Title	Credits	Type (T+P+Pj)
CUCS1008	Theory of Computation and Compiler Design	4	2+2+0

Course Description:

This course offers a comprehensive exploration of the theoretical foundations of computation and the principles of compiler design. Students will delve into the fundamental concepts that underpin computational theory and the mechanisms involved in translating high-level programming languages into machine code.

Course Objectives:

- To understand the theory of computation, basics of compiler design, and finite automata.
- To learn about lexical analysis, context-free grammars, syntax analysis, pushdown automata, and semantic analysis.
- To explore intermediate code generation, decidability, code optimization techniques, and develop compilers to solve complex computational problems.

Course Outcomes:

- **CO1**: Recall fundamental concepts of automata and compiler design. *(Remembering)*
- **CO2**: Explain principles of finite automata, context-free grammars, and Turing machines. *(Understanding)*
- **CO3**: Apply lexical and syntax analysis techniques. *(Applying)*
- **CO4**: Analyze various parsing and optimization techniques. *(Analyzing)*
- **CO5**: Develop components of a compiler using theoretical concepts. *(Creating)*

Course Outcome to Program Outcome Mapping:

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	1	1	-	-	-	-	-	-	-	-	-	2	1	1
CO2	3	2	1	-	-	-	-	-	-	-	-	-	2	1	1
CO3	3	3	2	-	-	1	-	-	-	-	-	-	2	1	1
CO4	3	3	2	1	-	1	-	-	-	-	-	-	3	2	1
CO5	3	3	3	2	-	1	-	-	-	-	-	-	3	3	2

***High-3, Medium-2, Low-1**

Course Syllabus:

Module 1: Foundations of Computation and Compiler Design (12 hours)

Theory

- Overview of Theory of Computation, Compiler Design
- Formal Languages, Automata, Compiler Structure, Phases

Practice

- Experiment 1.1: Design a DFA for a given language.
- Experiment 1.2: Implement an NFA and convert it to DFA.
- Experiment 1.3: Develop a regular expression for a given language.
- Experiment 1.4: Write a program to simulate a DFA.
- Experiment 1.5: Implement a lexical analyzer using Lex/Flex.
- Experiment 1.6: Develop a program to recognize tokens from a given input.

Module 2: Finite Automata and Lexical Analysis (12 hours)

Theory

- Deterministic Finite Automata (DFA), Non-Deterministic Finite Automata (NFA)
- Equivalence, Lexical Analysis, Token Specification, Lexical Analyzer Generator

Practice

- Experiment 2.1: Design a lexical analyzer for a subset of C language.
- Experiment 2.2: Implement a finite automaton for a given pattern.
- Experiment 2.3: Develop a lexical analyzer to count the number of tokens.
- Experiment 2.4: Write a program to simulate an NFA.
- Experiment 2.5: Implement a program to convert a regular expression to NFA.
- Experiment 2.6: Develop a program to minimize a DFA.

Module 3: Context-Free Grammars and Syntax Analysis (12 hours)

Theory

- Context-Free Grammars (CFG), Context-Free Languages (CFL)
- Derivations, Parse Trees, Ambiguity, Top-Down Parsing Techniques

Practice

- Experiment 3.1: Design a lexical analyzer for a subset of C language.
- Experiment 3.2: Implement a finite automaton for a given pattern.
- Experiment 3.3: Develop a lexical analyzer to count the number of tokens.
- Experiment 3.4: Write a program to simulate an NFA.
- Experiment 3.5: Implement a program to convert a regular expression to NFA.
- Experiment 3.6: Develop a program to minimize a DFA.

- **Module 4: Pushdown Automata and Bottom-Up Parsing**

(12 hours) Theory

- Pushdown Automata (PDA), Relation to CFGs
- Bottom-Up Parsing Techniques, Shift-Reduce Parsing, LR Parsing, Parser Generators

Practice

- Experiment 4.1: Design a PDA for a given language.
- Experiment 4.2: Implement a bottom-up parser for a given grammar.

- Experiment 4.3: Develop a program to simulate a PDA.
- Experiment 4.4: Write a program to implement shift-reduce parsing.
- Experiment 4.5: Implement a program to construct LR parsing table.
- Experiment 4.6: Develop a program to recognize context-free language.

Module 5: Turing Machines and Semantic Analysis (12 hours)

Theory

- Turing Machines, Decidable, Recognizable Languages, Halting Problem
- Syntax-Directed Definitions, Attribute Grammars, Type Checking, Symbol Tables

Practice

- Experiment 5.1: Design a Turing machine for a given language.
- Experiment 5.2: Implement a program to simulate a Turing machine.
- Experiment 5.3: Develop a program to perform type checking.
- Experiment 5.4: Write a program to implement a symbol table.
- Experiment 5.5: Implement a program to perform syntax-directed translation.

Module 6: Intermediate Code Generation and Decidability (12 hours)

Theory

- Intermediate Representations: Three-Address Code, Quadruples, Triples
- Syntax Tree, Directed Acyclic Graph (DAG)
- Decidable, Undecidable Problems, Halting Problem, Reductions

Practice

- Experiment 6.1: Write a program to generate three-address code.
- Experiment 6.2: Implement a program to generate quadruples.
- Experiment 6.3: Develop a program to generate triples.
- Experiment 6.4: Write a program to perform syntax tree construction.
- Experiment 6.5: Implement a program to solve halting problem.
- Experiment 6.6: Develop a program to perform code optimization.

Module 7: Code Optimization and Complexity Theory (12 hours)

Theory

- Code Optimization Techniques: Local, Global
- Time Complexity, P vs NP Problem, NP-Completeness, Data Flow Analysis, Peephole Optimization

Practice

- Experiment 7.1: Write a program to perform local optimization.
- Experiment 7.2: Implement a program to perform global optimization.
- Experiment 7.3: Develop a program to perform peephole optimization.
- Experiment 7.4: Write a program to analyze time complexity of an algorithm.
- Experiment 7.5: Implement a program to analyze space complexity of an algorithm.
- Experiment 7.6: Develop a program to solve P vs NP problem.

Text Books:

- John E Hopcroft, Rajeev Motwani, Jeffrey D. Ullman,“ Introduction to Automata Theory, Languages and Computation”, Third Edition, Pearson.
- Alfred V.Aho, Monica S.Lam,Ravi Sethi, Jeffrey D. Ullman, “ Compilers Principles, Techniques and Tools”, Second Edition,Perason.

Reference Books

- Elain Rich, “Automata, Computability and complexity”, 1st Edition, Pearson Education,2018.
- K.L.P Mishra, N Chandrashekaran , 3rd Edition , ‘Theory of Computer Science”,PHI,2012.