MCPE2006 COMPILER DESIGN (3-0-0)

Course Objectives: Upon successful completion of this course, students will be able to:

- Understand the Fundamental Principles and analyze Lexical Structure
- Implement Syntax Analysis, perform Semantic Analysis and design and generate suitable intermediate representations
- Apply Optimization Techniques and design Code Generators
- Utilize Compiler Construction Tools and develop a Foundational Understanding for Language Processors

Module I

Introduction, Phases of a compiler, Compiler construction tools. A simple one-pass compiler. Lexical Analysis: The role of the lexical analyzer. Input buffering. Specification of tokens: regular expressions. Recognition of tokens: finite automata (NFA and DFA). Conversion from regular expression to NFA. Conversion from NFA to DFA. Minimization of DFA. Design of a lexical analyzer generator (e.g., Lex/Flex concepts).

Module II

Introduction to Syntax Analysis, Role of the parser. Context-Free Grammars (CFG). Derivations, parse trees, ambiguity. Eliminating ambiguity. Top-Down Parsing, Recursive descent parsing. LL(1) grammars: conditions for LL(1). First and Follow sets. Construction of LL(1) parsing table. Error recovery in top-down parsing. Bottom-Up Parsing, Shift-reduce parsing. Handles and handle pruning. LR parsers: SLR, Canonical LR, LALR. Construction of LR parsing tables (conceptual overview for SLR).

Module III

Semantic Analysis: Syntax-Directed Translation (SDT): introduction to SDT schemes. Attribute grammars: synthesized attributes, inherited attributes. Dependency graphs. Type checking: type systems, static vs. dynamic checking. Symbol tables: structure, organization, operations. Run-time environment: storage organization, activation records, stack allocation. Intermediate Code Generation: Intermediate languages: three-address code (quadruples, triples, indirect triples). Postfix notation. Syntax trees.

Module IV

Code Optimization Introduction to optimization: principal sources of optimization.Basic blocks and flow graphs. Data flow analysis. Local optimizations: common subexpression elimination, dead code elimination, constant folding, strength reduction. Loop optimization: code motion, induction variable elimination. Peephole optimization. Code Generation: Issues in the design of a code generator. The target machine. Run-time storage management. Basic blocks and flow graphs (revisit). Simple code generator. Register allocation and assignment. Instruction selection.

Course Outcomes (COs):

Upon successful completion of this course, students will be able to:

CO 1: Analyze the phases of a compiler and design a lexical analyzer for a given programming language.
CO 2: Apply various parsing techniques to construct a parser for a given grammar.
CO 3: Develop syntax-directed translation schemes to perform semantic analysis and intermediate code generation.
CO 4: Apply different code optimization techniques to improve the efficiency of the generated code.

Text Book:

1. A.V. Aho, R. Sethi & J.D. Ullman “Compilers Principles Techniques and Tools” Pearson Education

Reference Books:

1. Kenneth C. Louden “Compiler Construction Principles & Practice “Cengage Learning Indian Edition