CSPC3004 COMPILER DESIGN (3-0-0)

Course Objectives

- To introduce the fundamental concepts of language processing and compiler design.
- To explain the design and implementation of the various phases of a compiler, including lexical, syntax, and semantic analysis.
- To equip students with knowledge of intermediate code generation, code optimization, and code generation techniques.
- To develop programming skills for building components of a compiler using tools like LEX and YACC.
- To explore runtime environment management and error handling in compiler design.

Course Outcomes

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

CO1: Understand the structure, phases, and functioning of a compiler.

CO2: Implement lexical analyzers and syntax analyzers using formal methods and tools.

CO3: Perform semantic analysis and generate intermediate code representations.

CO4: Design symbol tables, handle runtime environments, and implement error handling strategies.

CO5: Apply machine-independent and machine-dependent code optimization techniques and generate efficient target code.

Module I: (08 Hours)

Introduction to Compiler and Lexical Analysis

Language processing systems: preprocessors, compiler, assembler, interpreter, linker, bootstrap loaders, and cross compilers. Structure of a compiler and the different phases.

Lexical Analysis: Role of the lexical analyzer, input buffering, regular expressions, construction of NFA and DFA, DFA minimization, transition diagrams, token recognition. Lexical errors and recovery, introduction to lexical analyzer generator (LEX).

Module II: (08 Hours)

Syntax Analysis

Role of a parser, derivation, ambiguity, left recursion, left factoring, recursive descent parsing.

Bottom-up parsing: Shift-reduce parsing, operator precedence parsing, handles and handle pruning. Introduction to LR parsing: SLR, CLR, and LALR parsing tables. Parser conflicts and resolution, dangling else problem, error recovery strategies in parsing. Parser generator (YACC).

Module III: (08 Hours)

Semantic Analysis and Syntax-Directed Translation

Semantic Analysis: Syntax-directed definitions (SDD), evaluation of semantic rules.

Translation schemes: Syntax-directed translation schemes (SDTS), implementation of S-attributed and L-attributed definitions. Type checking and conversions.

Intermediate Code Generation: Abstract syntax trees, three-address code, quadruples, triples, indirect triples. Translation of expressions, assignment statements, Boolean expressions, case statements, back patching, and procedure calls.

Module IV: (08 Hours)

Symbol Table and Runtime Environment

Symbol table: structure, operations (insert, lookup), scope management, activation records.

Runtime environment: storage organization, stack allocation, heap management, access to non-local data, parameter passing mechanisms. Error handling: classification of errors, error recovery strategies.

Module V: (08 Hours)

Code Optimization and Code Generation

Code optimization:

- Machine-independent: constant folding, common subexpression elimination, dead code elimination, copy propagation, loop optimization, strength reduction, basic blocks and flow graphs, data flow analysis.
- Machine-dependent: DAG-based optimization, peephole optimization, register allocation, instruction scheduling, inter-procedural optimization, garbage collection using reference counting.
- Code generation: target machine model, issues in the design of a code generator, generation of code for expressions and control structures.

Textbooks

1. Compilers: Principles, Techniques, and Tools – A.V. Aho, M.S. Lam, R. Sethi, J.D. Ullman, Pearson Education, 2nd ed., 2007.
2. Principles of Compiler Design – V. Raghavan, Tata McGraw-Hill, 2nd ed., 2011.
3. Compiler Design in C – Allen I. Holub, Prentice Hall of India, 2003.

Reference Books

1. Compiler Construction: Principles and Practice – Kenneth C. Louden, CENGAGE.
2. Compiler Design – O.G. Kakde, University Science Press.
3. Compiler Design – K. Muneeswaram, Oxford University Press.