Compiler Design

L-T-P-C: 3-1-0-4

Course Objectives:

1. Students should learn how a high-level language (HLL) program is executed over a general purpose computer.

2. The two paradigms, compiling a HLL program to a low level language, and or to interpret the HLL program should become clear to students.

3. Students should become familiar with the principles, algorithms, and data structures involved in the design and construction of compilers.

Course Outcomes:

1. Various phases of a compiler viz., from lexical analysis to code generation should become familiar to the student.
2. The student should be able to work with tools like Lex and Yacc.
3. In an existing open source code of a compiler/interpreter the student should be able to understand various modules. And, as per the need the student should be able to rewrite some modules to suit the given task at hand.

Pre-requisites:

Theory of Computation, data structures and fundamentals of algorithms.

Syllabus:

Module 1:

Introduction – Various stages/phases – General applications of compiler building techniques.

Lexical Analysis – Need – Tokens, patterns, lexemes – Regular expressions, definitions – recognition – Lex tool – finite automata review (DFA, NFA, Minimization, etc) – symbol table.

Syntax Analysis – Need – Specification of syntax – CFG – Parsing, ambiguity – a review of PDAs – Eliminating left recursion – left factoring – Top-down parsing, recursive descent parsing, FIRST, FOLLOW, LL(1) grammars, LL(1) parsing, LL(1) parsing table, Nonrecursive predictive parsing.

Module 2:

Botton-up parsing, reductions, handles, shift-reduce parsing, conflicts, LR parsers, LR(0), SLR, CLR, LALR parsing techniques – Yacc tool.

Syntax-Directed Translation – Syntax directed definitions (SDD) – S-attributed, L-attributed SDD – DAG order of evaluation – Semantic rules, actions, actions within production – applications.

Intermediate-Code Generation – DAGs for expressions – Three-address code, quadruples, triples – Types, type expressions, equivalence, translation of type expressions – type checking – Control flow -- Intermediate code for procedures.

Module 3: Code (Machine-independent & dependent) optimization – source optimization – data-flow analysis – redundancy elimination – region based analysis – peephole optimization – optimal code for expressions.

Code generation – Instruction selection (tree rewriting), register allocation – Static allocation, stack allocation, runtime addresses – Basic blocks, flow graphs – register and address descriptors – an example case study.

Books:

1. "Compilers: Principles, Techniques, and Tools", Aho, Lam, Sethi, and Ulman.
2. "Engineering a compiler", Cooper and Torczon.
3. "The Compiler Design Handbook: Optimizations and Machine Code Generation", Y.N. Srikant and P.Shankar.
4. "Crafting Compiler", Fischer.