RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Computer Science & Information Technology, VII-Semester

Departmental Elective CSIT- 702 (B) Compiler Design

Course Objective:
The Objectives of this course is to explore the principles, algorithms, and data structures involved in the design and construction of compilers. Topics include context-free grammars, lexical analysis, parsing techniques, symbol tables, error recovery, code generation, and code optimization.

Course outcomes:
1. State the overview of phase of compiler and Lexical analysis.
2. Design and implement various parsing techniques of compiler.
3. Apply type checking for semantic analysis and analyze Run time environment.
4. Design and implement different intermediate code generation techniques.
5. Analyze various code optimization techniques.

UNIT I
Introduction to compiling & Lexical Analysis
Introduction of Compiler, Major data Structure in compiler, types of Compiler, Front-end and Back-end of compiler, Compiler structure: analysis-synthesis model of compilation, various phases of a compiler, Lexical analysis: Input buffering , Specification & Recognition of Tokens, Design of a Lexical Analyzer Generator, LEX.

UNIT II
Syntax Analysis &Syntax Directed Translation
Syntax analysis: CFGs, Top down parsing, Brute force approach, recursive descent parsing, transformation on the grammars, predictive parsing, bottom up parsing, operator precedence parsing, LR parsers (SLR,LALR, LR),Parser generation. Syntax directed definitions: Construction of Syntax trees, Bottom up evaluation of S-attributed definition, L-attribute definition, Top down translation, Bottom Up evaluation of inherited attributes Recursive Evaluation, Analysis of Syntax directed definition.

UNIT III
Type Checking & Run Time Environment
Type checking: type system, specification of simple type checker, equivalence of expression, types, type conversion, overloading of functions and operations, polymorphic functions. Run time Environment: storage organization, Storage allocation strategies, parameter passing, dynamic storage allocation , Symbol table, Error Detection & Recovery, Ad-Hoc and Systematic Methods.

UNIT IV
Code Generation
Intermediate code generation: Declarations, Assignment statements, Boolean expressions, Case statements, Back patching, Procedure calls Code Generation: Issues in the design of code generator, Basic block and flow graphs, Register allocation and assignment, DAG representation of basic blocks, peephole optimization, generating code from DAG.

UNIT V

Code Optimization

Introduction to Code optimization: sources of optimization of basic blocks, loops in flow graphs, dead code elimination, loop optimization, Introduction to global data flow analysis, Code Improving transformations ,Data flow analysis of structure flow graph Symbolic debugging of optimized code.

Recommended Books:

1. A. V. Aho, R. Sethi, and J. D. Ullman. Compilers: Principles, Techniques and Tools, Pearson Education
2 Raghavan, Compiler Design, TMH Pub.
3. Louden. Compiler Construction: Principles and Practice, Cengage Learning
4. A. C. Holub. Compiler Design in C , Prentice-Hall Inc., 1993.
5. Mak, writing compiler & Interpreters, Willey Pub.

List of Experiments:

1. Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.
2. Write a C program to identify whether a given line is a comment or not.
3. Write a C program to recognize strings under 'a*', 'a*b+', 'abb'.
4. Write a C program to test whether a given identifier is valid or not.
5. Write a LEX Program to count the number of token.
6. Write a LEX Program to identify the identifier.
7. Write a LEX Program to convert the substring abc to ABC from the given input string.
8. Write a lex program to find out total number of vowels, and consonants from the given input sting.
9. Write a C program to implement operator precedence parsing.
10. Write a C program to implement LALR parsing.