

Unit-I Introduction to compiling & Lexical Analysis

Introduction of Compiler, Major data Structure in compiler, BOOT Strapping & Porting, Compiler structure: analysis-synthesis model of compilation, various phases of a compiler, Lexical analysis: Input buffering , Specification & Recognition of Tokens, 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

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.

List of Experiments:

1. Develop a lexical analyzer to recognize a few patterns.
2. Write a programme to parse using Brute force technique of Topdown parsing.
3. Develop LL (1) parser (Construct parse table also).
4. Develop an operator precedence parser (Construct parse table also)
5. Develop a recursive descent parser
6. Write a program for generating for various intermediate code forms
i) Three address code ii) Polish notation
7. Write a program to simulate Heap storage allocation strategy
8. Generate Lexical analyzer using LEX
9. Generate YACC specification for a few syntactic categories.
10. Given any intermediate code form implement code optimization techniques
11. Study of an Object Oriented Compiler.

References:

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.