Compiler Design

Introduction & Pre-requisite

Dr. Mousumi Dutt

Referred Books

1. **Aho, Sethi, Ullman, Compiler Principles, Techniques and Tools**
2. Holub, Compiler Design in C
3. S. Chattopadhyay, Compiler Design (PHI)
4. C. Kenneth, Compiler Construction, Principles and Practice
5. Keith D. Cooper, Linda Torczon, Engineering a Compiler
6. Dick Grune, Kees van Reeuwijk, Henri E. Bal, Ceriel J.H. Jacobs, Koen Langendoen, Modern Compiler Design
7. A. Appel, Modern Compiler Imlementation in JAVA

Syllabus

- **Compiler Design**
- **CS702**
- **Contacts: 3L**
- **Credits- 3**

- **Prerequisite Subjects:**
- Formal Languages and Automata Theory
- Operating Systems, Computer Organization and Computer Architecture

Syllabus

- **Module I**
- **Introduction to Compiling** [2L]
- Compilers, Analysis-synthesis model , The phases of the compiler, Cousins of the compiler
- **Lexical Analysis** [5L]
- The role of the lexical analyzer, Tokens, Patterns, Lexemes, Input buffering, Specifications of a token, Recognition of tokens, Finite automata, From a regular expression to an NFA, From a regular expression to NFA, From a regular expression to DFA, Design of a lexical analyzer generator (Lex).

Syllabus

- **Module II**
- **Syntax Analysis** [8L]
- The role of a parser, Context free grammars, Writing a grammar, Top down Parsing, Non-recursive Predictive parsing (LL), Bottom up parsing, Handles, Viable prefixes, Operator precedence parsing, LR parsers (SLR, LALR), Parser generators (YACC). Error Recovery strategies for different parsing techniques.
- **Syntax directed translation** [4L]
- Syntax directed definitions, Construction of syntax trees, Bottom-up evaluation of S attributed definitions, L attributed definitions, Bottom-up evaluation of inherited attributes.

Syllabus

- **Module III**
- **Type checking** [3L]
- Type systems, Specification of a simple type checker, Equivalence of type expressions, Type conversions
- **Run time environments** [4L]
- Source language issues (Activation trees, Control stack, scope of declaration, Binding of names), Storage organization (Subdivision of run-time memory, Activation records), Storage allocation strategies, Parameter passing (call by value, call by reference, copy restore, call by name), Symbol tables, dynamic storage allocation techniques.

Syllabus

- **Module IV**
- **Intermediate code generation** [3L]
- Intermediate languages, Graphical representation, Three-address code, Implementation of three address statements (Quadruples, Triples, Indirect triples).
- **Code optimization** [4L]
- Introduction, Basic blocks & flow graphs, Transformation of basic blocks, Dag representation of basic blocks, The principle sources of optimization, Loops in flow graph, Peephole optimization.
- **Code generations** [3L]
- Issues in the design of code generator, a simple code generator, Register allocation & assignment.

About the Implementation

- **C programming language**
- **LEX tools - FLEX**
- **YACC tools - BISON**

Assessment

- **Initially weekly test or assignments**
- **Later programs will be given to implement**
- **Examinations as per instructions**

- **At the end of the course you will be able to design a compiler**

Course Outcome

The student will able to
CO1: Understand the basic concepts of language translation and compiler design.
CO.2: Apply the knowledge of lexical analysis
CO.3: Apply the knowledge of syntax analysis for designing compiler
CO.4: Design the next phases of compiler, i.e., semantic analysis and type checking
CO.5: Understand the issues with run time environments and develop intermediate code
CO.6: Perform the last phases of compiler design, i.e., code optimization and code generator

Compilers

- **A program that translates an executable program in one language into an executable program in another language**
- **The compiler typically lowers the level of abstraction of the program**
- **We expect the program produced by the compiler to be better than the original**

Importance of this subjects

- **In one sentence you will be able to design a compiler**

- **The following questions may arise**
  - **Why build compilers?**
  - **Why study compiler construction?**
  - **Why attend class?**

Reasons

- **Compilers provide an essential interface between applications and architectures**
- **Compilers embody a wide range of theoretical techniques**
- **Compiler construction teaches programming and software engineering skills**

Roles of Compilers- bridging gap

- High level programming languages
  - Increase programmer productivity
  - Better maintenance
  - Portable
- Low-level machine details
  - Instruction selection
  - Addressing modes
  - Pipelines
  - Registers and cache
  - Instruction-level parallelism

Compiler Construction:
Microcosmic View of Computer Science

artificial intelligence	greedy algorithms learning algorithms
algorithms	graph algorithms union-find network flows dynamic programming
theory	*dfa*'s for scanning parser generators lattice theory for analysis
systems	allocation and naming locality synchronization
architecture	pipeline management memory hierarchy management instruction set use

Enhance Software Engineering Skills

1. Correct code
2. Output runs fast
3. Compiler runs fast
4. Compile time proportional to program size
5. Support for separate compilation
6. Good diagnostics for syntax errors
7. Works well with the debugger
8. Good diagnostics for flow anomalies
9. Cross language calls
10. Consistent, predictable optimization

More on next class