

COURSE DESCRIPTION

Department and Course Number: COMP 6210

Course Title: Compiler Construction

Total Credits: 3

Required: No

Prerequisites: COMP 3220 and COMP 4200

Class meetings per week: 3 hours

Lab meetings per week: 0 hours

Course Coordinator: Dr. Min-Te Sun

Date Prepared: February 18, 2004

Current Catalog Description:

Compiler organization; lexical analysis; parsing; syntax-direction translation; symbol tables; basic dependence analysis; intermediate forms; interpreters vs. compilers; run-time storage management; code generation; error detection and recovery.

Textbooks:

D. Grune, H. Bal, C. Jacobs, and K. Langendoen. 2000. *Modern Compiler Design*. John Wiley & Sons. ISBN 0471113530.

References:

Lex and Yacc. Levine, Mason, and Brown, O'Reilly, 1992.

Course Objectives:

1. Be able to understand and develop a compiler for an imperative high-level language.
2. Be able to understand the design trade-offs involved in each phase of compilation: lexical analysis, parsing, intermediate form, and code generation.

Prerequisites by Topic:

1. Knowledge of formal languages and various computational models, including regular expressions, finite automata, context-free grammars, parsing, and ambiguity
2. Familiarity with at least one high-level programming language, such as C, C++, or Java

Topics Covered: (specify number of hours on each)

1. Structure of a compiler (2 hours)
2. Language translation: lambda calculus, the SECD machine (5 hours)
3. Simple compiler (3 hours)
4. Lexical analysis (2 hours)
5. Generating a lexical analyzer from regular expressions (2 hours)
6. Top-down parsing (3 hours)
7. Bottom-up parsing (4 hours)
8. Generating a bottom-up parser from a context-free grammar (3 hours)
9. Symbol tables (2 hours)
10. Intermediate forms (2 hours)

11. Dependence analysis (2 hours)
12. Optimization (2 hours)
13. Run-time storage management (6 hours)
14. Code generation (5 hours)
15. Exams (2 hours)

Laboratory Projects: (specify number of weeks on each)

1. Lexical analyzer (2 weeks)
2. Syntax analyzer (2 weeks)
3. Semantic analyzer (2 weeks)
4. Intermediate code generator (2 weeks)
5. Code optimizer (2 weeks)
6. Code generator (2 weeks)
7. Compiler constructed from the previous small projects (2 weeks)

Oral and Written Communications:

Students are grouped into teams to work on the programming projects. The project requires students to develop and apply program documentation skills as part of the project requirement. At the end of the semester each team will held a short oral presentation to demonstrate the compiler constructed from small projects.

Social and Ethical Issues:

None.

Theoretical Content:

Fundamentals of formal languages are presented in focused lessons (3 hours) and then applied as appropriate throughout the course. This theoretical content is review only since it was covered in-depth by the prerequisite courses.

Problem Analysis and Solution Design:

All students apply fundamental software engineering practices to analyze, design, implement, test, and document each small project that aims at final compiler construction. Students apply the analysis and design skills already acquired to the development of software components that implement parts of a compiler. Each component has stated requirements and students are responsible for applying a controlled, iterative process for development. The requirements for components non-trivial and involve significant problem analysis and solution design. Students are required to integrate the components to form a single, working compilation system.

