

CMPSC 471

Introduction to Compiler Construction

Catalog Data: Introduction to Programming Techniques (3)
Design and implementation of compilers; lexical analysis, parsing, semantic actions, optimization, and code generation. Prerequisite: CMPSC 461.

Typical Textbook: *Engineering a Compiler*, by K. Cooper and L. Torczon. Published by Morgan-Kaufmann, 2004. Required.

Course Objectives: CMPSC 471 introduces the fundamental concepts in compiler design. The topics covered include scanner and parser designs and implementation, program shape analysis, intermediate code generation, and back-end optimizations such as instruction selection and scheduling. The goal is to familiarize students with basic structure of a typical modern compiler and help them implement several sample compiler phases (passes) – typically one from front-end and one from back-end. Another goal is to help them understand the implementation consequences of the choices made in programming language design.

Primary Course Outcomes: Upon completion of the course, students should possess the following skills:

- **Compiler Design:** Given a basic compiler definition, students can decide the corresponding compiler structure and identify the relationships among different phases of the compiler.
- **Compiler Implementation:** Given a compiler design, students can implement required modules, which may include front-end, back-end, and a small set of middle-end optimizations.
- **Documentation:** Students can document the compiler structure and phases which conforms to the underlying programming language specifications.
- **Cost Analysis:** Given a high-level programming language construct, students can analyze and quantify (where appropriate) different potential implementations of the construct from the performance and space requirement viewpoints.
- **Testing and Debugging:** During compiler development process, students can utilize basic testing methodologies and debugging tools such as stubs, drivers, and integrated debuggers to identify fault points and possible error conditions, and exceptions with try-catch blocks.
- **Broader Application:** Students can apply classical compilation techniques to diverse data processing applications such as word processors and workflow managers.

Relationship to Undergraduate Program Outcomes:

- Analyze algorithms or computer code for correctness and efficiency.
- Develop a modest (on the order of a thousand lines of code) software application, using
 - appropriate data structures and algorithms.
 - Write clear and effective technical prose.
 - Be able to discuss major trends in industry and current research activities within the discipline.

Required Topics: Introduction, overview of programming languages and constructs (~1.5 hours)
Regular expressions and scanner design (~3 hours)
Top-down and bottom-up parser designs (~6 hours)
Attribute grammars and syntax-directed translation (~3 hours)
Intermediate representations and symbol tables (~1.5 hours)
Memory management (~1.5 hours)
Code shape analysis and procedure abstraction (~6 hours)
Code generation and back-end optimizations (~7.5 hours)

Introduction to middle-end optimizations and dataflow analysis (~3 hours)
Dynamic compilation (~1.5 hours)

Class Format: Two lecture/labs per week. Each lecture/lab is 75 minutes.

Professional Component: CMPSC 471 is designed to aid in professional development of engineers and scientists by developing skills in solving several challenging optimization problems. These problems find their roots in formal computational theory, graph theory, and linear algebra. Many of the issues covered in the class will help students to tackle other (similar) problems they will face in large scale software design and maintenance.

Evaluation: 50 – 70 % proctored assessments (exams/labs)
30 – 50 % unproctored assignments (programming projects, take-home labs, etc)

Suggested breakdown based on 1000 pts as follows:

250 Midterm
350 Final
200 Homeworks
200 Programming Projects

Programming Assignments: Tentative Schedule

| Goals | Assigned | Due | Worth |
|---|-----------------|------------|--------------|
| Parser Design and Implementation | L7 | L12 | 30% |
| Instruction Scheduler Design and Implementation | L13 | L19 | 35% |
| Register Allocators Design and Implementation | L20 | L28 | 35% |

Author: Mahmut Kandemir
Last Revised: February 2008