COMP 412
Overview of the Course

Critical Facts

COMP 412 — Introduction to Compiler Construction

Topics in the design of programming language translators, including parsing, run-time storage management, error recovery, code generation, and optimization

- Instructor: Keith Cooper <keith@rice.edu>
- Office Hours: TBD, DH 2065 or DH 3131
- Text: Engineering a Compiler (Cooper and Torczon)
  - Published by Morgan-Kaufmann
  - Royalties for sales to COMP 412 go to the Torczon Fellowship at Rice, which awards a fellowship to a Rice CS undergrad
- Web Site: http://owlnet.rice.edu/~comp412
  - Lab handouts, homework, slides, practice exams, …
  - I will not have handouts in class; get them from the web

Basis for Grading

- Exams
  - Midterm 25%
  - Final 25%
- Homework 5%
- Projects
  - Register allocator 15%
  - Parser (& scanner) 15%
  - Instruction scheduler 15%

This stuff will total 55% of the course credit. If I assign more homework than in the past, I may shift the percentages by up to 5%

Notice: *Any student with a disability requiring accommodations in this class is encouraged to contact me after class or during office hours, and to contact Rice's Coordinator for Disabled Student Services.*

Basis for Grading

- Exams
  - Midterm
  - Final
  - ✦ Closed-notes, closed-book take home
  - ✦ Old exam on web site as an example
- Homework
  - ✦ Reinforce concepts, provide practice
  - ✦ Number of assignments *t.b.d.*
- Projects
  - Register allocator
  - Parser (& scanner)
  - Instruction scheduler
  - ✦ High ratio of thought to programming
  - ✦ Parser lab might be a team lab
  - ✦ Choose your own language (*not* ***PERL***)

Rough Syllabus

• Overview	§ 1
• Local Register Allocation	§ 13.2
• Scanning	§ 2
• Parsing	§ 3
• Context Sensitive Analysis	§ 4
• Inner Workings of Compiled Code	§ 6, 7
• Introduction to Optimization	§ 8
• Code Selection	§ 11
• Instruction Scheduling	§ 12
• Register Allocation	§ 13
• More Optimization (*time permitting*)	

If it looks like the course follows the text, that's because the text was written from the course.

What about the missing chapters?

5 : We'll fit it in

9, 10: see CS 512

Class-taking technique for COMP 412

- I will use projected material extensively
  - But I will try to stimulate discussion
- Read the book
  - Not all material will be covered in class
- Come to class
  - The tests will cover both lecture and reading
  - I will probably hint at good test questions in class
- Do the programming assignments
  - COMP 412 is not a programming course
    - Projects are graded on functionality, documentation, and lab reports, not style (*results matter*)
- Do the homework
  - Good practice for the tests

About the Book

- Textbook: "Engineering a Compiler"
  - By Keith D. Cooper and Linda M. Torczon
    - Both at Rice
- Book presents modern material
  - Considers problems of post-1986 computers
  - Addresses modern techniques
  - Discards lots of less relevant material
- Other textbooks on reserve in Fondren Library
  - Consult them for alternate views

Compilers

- What is a compiler?
  - A program that translates an *executable* program in one language into an *executable* program in another language
  - The compiler should improve the program, *in some way*
- What is an interpreter?
  - A program that reads an *executable* program and produces the results of executing that program
- C is typically compiled, Scheme is typically interpreted
- Java is compiled to bytecodes (code for the Java VM)
  - which are then interpreted
  - Or a hybrid strategy is used
    - Just-in-time compilation

Taking a Broader View

- Compiler Technology = Off-Line Processing
  - Goals: improved performance and language usability
    - Making it practical to use the full power of the language
  - Trade-off: preprocessing time versus execution time (or space)
  - Rule: performance of both compiler and application must be acceptable to the end user
- Examples
  - Macro expansion
    - PL/I macro facility — 10x improvement with compilation
  - Database query optimization
  - Emulation acceleration
    - TransMeta "code morphing"

Why Study Compilation?

- Compilers are important
  - Responsible for many aspects of system performance
- Compilers are interesting
  - Compilers include many applications of theory to practice
  - Writing a compiler exposes practical algorithmic & engineering issues
- Compilers are everywhere
  - Many practical applications have embedded languages
    - Commands, macros, formatting tags …
  - Many applications have input formats that look like languages

Intrinsic merit

- Compiler construction poses challenging and interesting problems:
  - Compilers must do a lot but also run quickly
  - Compilers have primary responsibility for run-time performance
  - Compilers are responsible for making it acceptable to use the full power of the programming language
  - Computer architects perpetually create new challenges for the compiler by building more complex machines
    - Compilers must hide that complexity from the programmer
- A successful compiler requires mastery of the many complex intereactions between its constituent parts

Making Languages Usable

It was our belief that if FORTRAN, during its first months, were to translate any reasonable "scientific" source program into an object program only half as fast as its hand-coded counterpart, then acceptance of our system would be in serious danger... I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed.

— John Backus

Intrinsic interest

- Compiler construction involves ideas from many different parts of computer science

Artificial intelligence	Greedy algorithms Heuristic search techniques
Algorithms	Graph algorithms, union-find Dynamic programming
Theory	DFAs & PDAs, pattern matching Fixed-point algorithms
Systems	Allocation & naming, Synchronization, locality
Architecture	Pipeline & hierarchy management Instruction set use

About the Instructor

My own research program

- Compiling for advanced microprocessor systems
  - Optimization for *space, power, & speed*
- Whole program analysis & optimization
- Relationship between compiler structure & effectiveness
- Nitty-gritty things that happen in compiler back ends

Thus, my interests lie in

- Quality of generated code
- Interplay between compiler and architecture
- Static analysis to discern program behavior
- Run-time performance analysis

Next class

- The view from 35,000 feet
  - How a compiler works
  - What is important
  - What is hard and what is easy

- Things to do
  - Make sure you have a working OwlNet account
  - Find the web site

www.owlnet.rice.edu/~comp412