15-411 Compiler Design

Fall 2014 / Frank Pfenning

Teaching Staff

- Instructor
  - Frank Pfenning, GHC 7019
  - Office hour: Thu 10:30-12:00
- Teaching Assistants (GHC 5205 for now)
  - Flávio Cruz, Thu 1:30-3:30 (starting Sep 10)
  - Max Serrano, Wed 4:00-6:15
  - Rokhini Prabhu, Mon 6:00-8:00
  - Tae Gyun Kim, Tue 4:30-6:30
- Updates on Piazza

Course Communication

- Lectures: Tue & Thu, 9:00-10:20, WeH 7500
- Recitations: none
- Piazza (including partner search)
  - You will be enrolled with your Andrew ID
- Autolab
  - You will be enrolled with your Andrew ID
  - Hand-in for labs
  - Maintains course grades
- http://www.cs.cmu.edu/~fp/courses/15411-f14

Learning Goals: Compilers

- Compilers: from program text to machine code
- The structure of compilers
- Applied algorithms and data structures
  - Context-free grammars and parsing
  - Static single assignment form
  - Data flow analysis
  - Chordal graph coloring
- Focus on sequential imperative programming language
  - Not functional, parallel, distributed, object-oriented, …
- Focus on code generation and optimization
  - Not error messages, type inference, runtime system, …

Learning Goals: Software Engineering

- A compiler is a substantial piece of software
  - Building, testing, debugging, evolving
  - Solo, or in a team of two
- Understanding high-level specifications
- Satisfying performance constraints
- Making and revising design decisions
  - Implementation language
  - Data representation
  - Algorithms
  - Modules and interfaces

Role in the Curriculum

- 15-213 Introduction to Computer Systems
  - Prerequisite
- 15-411 Compiler Design
  - How are your high-level programs translated to low level?
- 15-410 Operating System Design and Implementation
  - How is the execution of your programs managed?
- 15-441 Computer Networks
  - How do programs communicate?
- 410, 411, 441 all satisfy system requirement
- 15-417 HOT Compilation
  - How to compile higher-order typed languages

Course Materials

- Extensive lecture notes
  - Usually out a few days after lecture
- Textbook (optional)
  - Andrew Appel, *Modern Compiler Implementation in ML*
- Lab specifications
  - Details of language fragments you implement

Your Responsibilities

- Lectures
  - Lecture notes and readings only supplement lecture
- 5 written homeworks (30% of grade)
  - Done individually
- 6 labs
  - Done individually or in pairs
  - Labs 1-4 (40% of grade)
    - Write complete compilers and tests for increasingly complex languages
  - Lab 5-6: (30% of grade)
    - Extend in a direction that interests you; submit two papers
- No midterm exam, no final exam
- Academic integrity policy applies

Homeworks

- Prepares you for lab
- 5 homeworks, about one week each
- Must be your own work
- 30% of final grade
- Due at beginning of lecture
    - Up to two homeworks can be late, max of 2 days

Labs – Overview

- Submitted through Autolab
  - Week 1: test cases (validated against reference compiler)
  - Week 2: compilers (checked against all test cases)
- Must be entirely your team's work
  - Acknowledge sources in readme.txt
  - Can also be done individually, but less fun
- Autograded
  - Against everyone's test cases
  - From this year and last year and …
  - Reserve the right to inspect code
  - Usually, feedback on code only if requested

Labs – Language(s) to Compile

- C0, a small safe subset of C
  - Designed by me and collaborators for teaching imperative programming at the freshman level (15-122)
  - Small
  - Safe
  - Fully specified
  - Augmented by a layer of contracts
- Rich enough to be representative and interesting
- Small enough to manage in a semester
- Use student compiler from 15-411 in 15-122 (some day)
  - Or at least the code generator

Labs – Language(s) to Target

- x86_64 architecture
  - Widely used
  - Quirky, but you can choose the instructions you use
  - Low level enough you can "taste" the hardware
- Runtime system
  - C0 uses the ABI (Application Binary Interface) for C
  - Strict adherence (internally, and for library functions)
- Similar to x86, different from ARM
- May retarget your compiler in Lab 6
  - LLVM (Low Level Virtual Machine)
  - ARM ?

Labs – Cumulative Compiler

- Cumulatively build a compiler for C0
  1. Expressions
  2. Control flow
  3. Functions
  4. Structs and arrays
  5. Optimizations (code + paper)
- Each one is a complete, end-to-end compiler
- Lab 6 open-ended, submit code + term paper
  - Retarget compiler
  - Garbage collector
  - Choose your own adventure

Labs – Implementation Language

- Choose your own implementation language
- Starter code for Lab 1 in
  - SML
  - Haskell
  - Scala (no longer supported)
  - Java
  - O'Caml
  - C++??

Labs – Submission

- SVN repositories set up for each group
- 'svn update'
  - For starter code in Lab 1
  - For lab specification
  - For test cases
  - For runtime system and grading script
- 'svn commit'
- Ask to be checked out and graded in Autolab
- Late days
  - 6 total for semester
  - At most 2 per lab
  - Don't fall behind!

Labs – Partners

- Use Piazza to choose partners (if needed)
- Each one is responsible for all code
  - Swap roles between labs
  - Both must pull their weight
- Should decide by week 2
- Contact instructor if you have problems

Labs – Advice

- Labs are difficult and take time
  - Plan ahead!
  - Set up meetings with lab partners
  - Talk to us and others about design decisions
- Don’t start the compiler only after the tests
- Errors in lab carry over to next lab
- Compilers are complex artifacts
  - That’s one thing that makes them fun
  - Hone your software engineering skills
- Submit early and often