Course Introduction

Dr. Mattox Beckman

Illinois Institute of Technology
Department of Computer Science

Welcome to CS 443!

Topics for discussion:

- Logisitics — instructor, grades, course objectives, lecture format
- What is a Compiler?

Me!

Name	Mattox Beckman
History	PhD, Fall 2003, University of Illinois at Urbana-Champaign
Research Areas	Programming Languages, Mathematical Foundations of Computer Science
Specialty	Partial Evaluation, Functional Programming
Professional Interests	Teaching; Partial Evaluation; Interpreters; Functional Programming; Semantics and Types; Category Theory
Personal Interests	Cooking; Go (Baduk, Wei-Qi, Igo); Theology; Evolution; Greek; Meditation; Kerbal Space Program; Home-brewing; ... and many many more ...

Teaching philosophy is available at
http://mccarthy.cs.iit.edu/mattox/static/teaching-philosophy.pdf

My Responsibilities

My job is to provide an "optimal learning environment".

- Assignments will be clearly written and administered.
- Questions will be answered in a timely fashion.
- Objectives of lectures and assignments will be clearly communicated.
- Grades will be fair, meaningful, and reflect mastery of course material.
  - C grade means "can reliably recognize the correct answer"
  - A grade means "can reliably generate the correct answer"
- **If something's not going the way it should, tell me!**

Your Responsibilities

- Check the course web page frequently.
  `http://mccarthy.cs.iit.edu/cs443`
- Subscribe to Piazza and have at least digest email.
- Do the homework assignments in order to learn them.
- Attend lectures if at all possible.
- **Take responsibility and initiative in learning material** — experiment!

You are the one primarily responsible for your education.

Contact Info

Instructor Mattox Beckman

Best Contact via email. I use inbox zero, but not on weekends.

Email Addresses <beckman@iit.edu> or <mattoxbeckman@gmail.com> They go to the same inbox. Don't spam them.

Office 110 SB

Office Hours 10:30–11:30 M,T

Home Page http://mccarthy.cs.iit.edu/mattox

Teaching Assistant Keep dreaming.

Machine Problems and Activities

- Machine Problems — collectively worth 30%
- This is your project. The MPs serve as “checkpoints”
- Up to three people may work together.
- The in-class activities will be MP-like and collectively be worth 10%.

Exams

- The purpose of an exam is to measure mastery of material.
- 2 midterm exams: 20% each.
  - It will be held during class.

    Dates Wednesday, February 25; Wednesday, April 8
- Final exam: 20%
  - Date: To be decided
  - Cumulative
  - Nice British System

Grade Guarantees

The course will not be graded on a curve or by ranking. Instead, we have the following grade guarantees:

- 85% A
- 70% B
- 55% C
- 40% D

What is a compiler?

- Translation — usually (but not necessarily!) to a lower level language.
- Usually different language families, and should bring about some form of improvement (or else why bother)?
- Tend to be very complex, multi-stage creatures.

Part 1 — Parsing

- We discussed these in 440; we will implement one in 443.
- Purpose: to translate text (ASCII or Unicode) to Abstract Syntax Tree
- Usually do some semantic analysis here as well.

Part 2 — Semantic analysis

- Create a symbol table for function and variable (and type) declarations
- Check to see that the types are correct.
- Catch “obvious” errors.
- We might do stack frame things here too.

Part 3 — Generate Intermediate Code

- Canonicalize the source language first (usually)
- Convert to a simpler language
  - Complex enough to be easy to translate to
  - Simple enough to convert to assembly
  - This also helps modularize the compiler

Part 4 — Basic Block Analysis

- Basic blocks have one entry point and one exit point.
- They can be combined.
- This groups instructions together to form larger units for analysis.

Part 5 — Instruction Selection

- It's starting to look like machine code!
- We pretend we have an infinite number of registers.
- Sometimes there are multiple ways to select the instructions...
- Use greedy algorithms or dynamic programming!

Part 6 — Liveness Analysis

- A variable is rarely needed for as long as it exists.
- We can use one register to represent multiple variables!
- We use a flow propagation algorithm to generate this.
- Used a preparation for register allocation.

Part 7 — Register Allocation

- Replace the infinite number of pseudo-registers with real registers.
- Uses graph coloring.
- Sometimes it doesn't work: we need to use memory instead.

Final notes

- This is how a "traditional" compiler is set up.
- We may be able to talk about HASKELL/Core style compilers.
- You might not need full compilation though!
  - Source-to-source translation. Just compile to C and let the C compiler do the work!
  - Macros: for compiling DSLs.