Why Is Programming So Hard?

Software is different from other artifacts
- We build general, reusable mechanisms
- Not much repetition, symmetry, or redundancy
- Large systems have millions of distinct complex parts

Goals

- Primary focus: writing correct programs
  - What does it mean for a program to be correct?
  - How do we determine if a program is correct?
  - How do we build correct programs?
- Will cover both *principles* and tools.
- Tools change, principles are forever...

“Controlling complexity is the essence of computer programming.”

-- Brian Kernighan
(UNIX, AWK, C, ...)
Outcomes

- Better at design
- Better at writing
- Better at debugging
- Better at using development tools
- Better at evaluating quality / behavior
- Better at communication
  - Can you convince yourself and others something is correct via precise, coherent explanations?
- Essential skills regardless of what you do next

A Problem

“Complete this method so that it returns the index of the max of the first \( n \) elements of the array \( a \).”

```swift
func indexOfMax(a: [Int], n: Int) -> Int {
…
}
```

A Problem

“Complete this method so that it returns the index of the max of the first \( n \) elements of the array \( a \).”

```swift
func indexOfMax(a: [Int], n: Int) -> Int {
…
}
```

Prerequisites

- Proficient in Java, eg:
  - Sharing:
    - Distinction between == and equals()
    - Aliasing: multiple references to the same object
  - Object-oriented dispatch:
    - Inheritance and overriding
    - Objects/values have a run-time type
  - Subtyping
    - Expressions have a compile-time type
    - classes vs. interfaces
- Reasoning and proof techniques
- Basic Unix and OS X skills
Course Components

• Lecture
• Reading
• Written Homework
• Labs
• Final Project
• Exams

• CS 326 Web Page
• Honor Code

Text Resources

• The Pragmatic Programmer
  – Hunt and Thomas (1999)
  – Collection of best practices

• Class notes, additional readings

• Swift Language and API Docs:
  – Swift Language

Pragmatic Programmer

• Advice from top-notch programmers
• Stuff all serious programmers should know
• Approachable but sometimes challenging
• Only partial overlap with lecture

• Keep up with reading
  – Reading and contemplating design is essential
  – Time investment that pays dividends in the long run

Programming is Hard

• Despite decades of practice, still surprisingly difficult to specify, design, implement, test, and maintain even small, simple programs.

• Assignments will be reasonable if you apply the techniques taught in class...
  ... but likely very difficult to do brute-force ...
  ... and almost certainly impossible unless you start early.

• Think before you type!
Looking Ahead A Few Weeks

You Have Lab Today!

- **Lab 0**
  - Please do first step before noon today!
- Set up lab environment
- Git
- Markdown
- Swift Tutorial

You Have Homework For Tues.

- **HW 1**
- Design algorithm to meet a simple specification
- Working up to reasoning about large designs

Motivation/Structure of CS 326

- My own experiences
  - 25+ years of building systems (successes/failures)
  - My research on languages and defect detection
- Hard work, course development, and insights of many others
  - Michael Ernst, Hal Perkins, Dan Grossman, David Notkin, Zach Tatlock, Paul Hegarty, Scott Smith