Introduction to Programming Systems

CS 217, Fall 2004

Randy Wang
Princeton University

Outline

- Administrative trivia
- Goals of the class
- Introduction to C

Precepts

- No precepts this (first) week
- Location TBA
- Assignment 1 available at the end of today

Goals

- Master the art of programming
  - Learn how to be “good” programmers
  - Introduction to software engineering
- Learn languages for systems programming
  - C is the systems language of choice
  - Assembly is required for low-level system programming
- Introduction to computer systems
  - Machine architecture
  - Operating systems
  - Software tools

Outline

- First three weeks
  - C programming language
- Next two weeks
  - Software engineering
- Next two weeks
  - Machine architecture
- Next two weeks
  - Software tools
- Next three weeks
  - Unix operating system services

Coursework

- Six programming assignments (60%)
  - Un-comment filter
  - String library
  - Hash table ADT
  - IA32 assembly language programming
  - IA32 assembler
  - Shell
- Exams (30%)
  - Midterm
  - Final
- Class participation (10%)

Assignments

- 1st assignment available at the end of today
- One “free” extension
  - Max of three days
  - Need to tell us when you want to use it
  - No other extensions (except for illness etc.)
- Read the “policy” page on the web
  - Pay special attention to collaboration policy

Materials

- Required textbooks
  - *C Programming: A Modern Approach,* King, 1996.
  - *The Practice of Programming,* Kernighan and Pike, 1999.
  - *Programming from the Ground Up (online)*, Bartlett 2004.
- Recommended textbooks
  - Programming with GNU Software. Loukides & Oram
- Other textbooks (on reserve)
  - IA32 Intel Architecture Software Developer's Manual (online)
  - The C Programming Language, Kernighan & Ritchie
  - C: A Reference Manual. Harbison & Steele
  - C Interfaces and Implementations. Hanson
  - The UNIX Programming Environment. Kernighan & Pike
- Web pages
  - **http://www.cs.princeton.edu/courses/archive/fall04/cos217/**

Facilities

- Unix machines
  - CIT's **`arizona (phoenix)`** cluster (Sparc)
  - OIT's **`hats`** cluster (Linux)
- Your own laptop
  - **`ssh`** access to **`arizona (or phoenix) and hats`**
  - run GNU tools on Windows
  - run GNU tools on Linux

Logistics

- Lectures
  - Introduce concepts
  - Work through programming examples
- Precepts
  - Review concepts
  - Demonstrate tools (gdb, makefiles, emacs, …)
  - Work through programming examples

Outline

- Administrative trivia
- Goals of the class
- Introduction to C

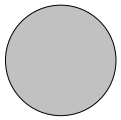
Software is Hard

"What were the lessons I learned from so many years of intensive work on the practical problem of setting type by computer? One of the most important lessons, perhaps, is the fact that SOFTWARE IS HARD. From now on I shall have significantly greater respect for every successful software tool that I encounter. During the past decade I was surprised to learn that the writing of programs for TeX and Metafont proved to be much more difficult than all the other things I had done (like proving theorems or writing books). The creation of good software demands a significantly higher standard of accuracy than those other things do, and it requires a longer attention span than other intellectual tasks."

Donald Knuth, 1989

The Discipline of Software Engineering

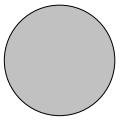
The beginning



Doesn't Work

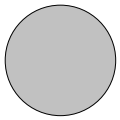
The Discipline of Software Engineering

The beginning



Doesn't Work

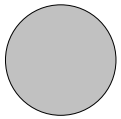
The hope



Works

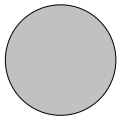
The Discipline of Software Engineering

The beginning



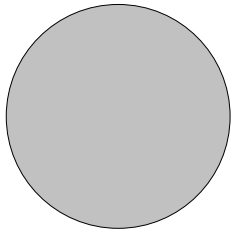
Doesn't Work

The hope



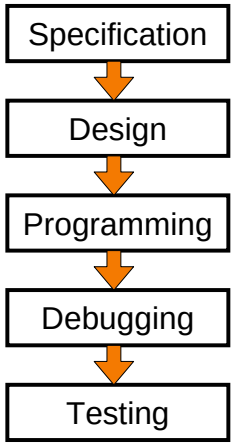
Works

The reality



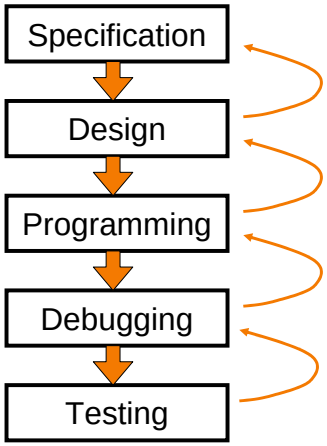
Still doesn't Work

Software in COS126



1 Person
10^2 Lines of Code
1 Type of Machine
0 Modifications
1 Week

Software in the Real World



Lots of People
10^6 Lines of Code
Lots of Machines
Lots of Modifications
1 Decade or more

Good Software in the Real World

- Understandable
  - Well-designed
  - Consistent
  - Documented

  ← Write code in modules with well-defined interfaces
- Robust
  - Works for any input
  - Tested

  ← Write code in modules and test them separately
- Reusable
  - Components

  ← Write code in modules that can be used elsewhere
- Efficient
  - Only matters for 1%

  ← Write code in modules and optimize the slow ones

Good Software in the Real World

- Understandable
  - Well-designed
  - Consistent
  - Documented

  ← Write code in modules with well-defined interfaces
- Robust
  - Works for any input
  - Tested

  ← Write code in modules and test them separately
- Reusable
  - Components

  ← Write code in modules that can be used elsewhere
- Efficient
  - Only matters for 1%

  ← Write code in modules and optimize the slow ones