CSE 142

Introduction to Programming

Overview

- Topics
  - Computers and Programming
  - Thinking about programs
- Reading
  - Dugan notes, ch. 1-3
  - Niño & Hosch, ch. 1 (skim)

Hardware and Software

- Hardware: the physical machine
  - CPU/processor, e.g. "Pentium"
  - Memory/RAM
  - Hard disk, floppy disk, CDROM, ...
  - Monitor, speakers, keyboard, mouse, ...
  - Network connection
- Hardware doesn't do anything on its own; no personality
- Software, i.e., programs: electronic plans that instruct the hardware how it should act; the personality
  - Easy to change software, if we have good programmers!

Learning Programming

- Programming is both easier and harder than most people make it out to be.
  - Easier: Many of the things good programmers do well are actually things all of us already do all the time, we just don't know it.
  - Harder: Programming is in large part a *skill*, even an *art*
- Programming is like any craft: it requires practice.
  - Learning by doing vs. learning by reading about it
  - Experiment: Not sure how something works? Try it and see.
  - Don't be afraid to build things that you'll throw away later.

Programming as Communication

- With a program, we communicate with two important entities:
  - machines
  - people
- The first is obvious
- The second may not be:
  - Programs that don't work (bugs)
  - Inherited code
  - Program evolution

Reading vs. Understanding

- People and machines are very different.
  - Machines are good are *reading* (details) but bad at *understanding* (what is intended)
  - People are good at *understanding* but bad at *reading*
- Read this:

Communicating with Computers and People

- Computers demand precision, logical thinking
  - Being precise, complete, and logical (thinking like a machine) is one thing that makes programming hard
  - Computers offer speedy, mistake-free results
- People can fill in missing steps, but can get swamped by lots of unorganized details and clutter
  - Need to write programs so that can be understood by people, e.g., your coworkers, your clients, yourself 3 months from now
  - Invent *abstractions*: new vocabulary, short-hands
  - Be *organized*, use good *style*

Example: Giving Directions

- Imagine giving campus directions:
  - To another student
  - To a tourist
  - To a robot
- The student operates at a higher level of *abstraction* with a richer *vocabulary* of *short-hands*
- An *algorithm* is a plan for how to accomplish a task
  - A *program* is a software implementation of an algorithm
- Good algorithms (at any level of abstraction) require precision

Metaphor: Programs as Directions

- One way to think about programming: a program is a sequence of commands that brings about some action.
- E.g. telling a robot how to navigate around campus

Metaphor: Programs as Math

- We also can think of programs as *executable math*: a program calculates some result for us.
- Consider:

$$\text{Area} = \text{PI} \times \text{Radius}^2$$

- We can employ such expressions in programs.
- Most of our intuitions and knowledge about mathematics apply to computers.
- Programs can compute more interesting things than just numbers, though
  - Sound, graphics, text, ...

Metaphor: Programs as Simulations

- We also can think of programming as creating or simulating both real *and virtual* worlds.
- We can define things in our programs that model the things in our world. We call these things *objects*.
- Programs are *plastic*: they are easy to mold to our wishes
  - Can be free of the constraints of real life!
- The limit of plasticity: big programs become as hard to work with as real-world entities