Computer Science 50
Fall 2011
Scribe Notes

Week 0 Friday: September 2, 2011
Andrew Sellergren

Contents

1 Announcements and Demos (0:00–15:00) **2**

2 Computer Science in the Real World (15:00–25:00) **3**

3 From Last Time (25:00–30:00) **3**

4 Binary and Hard Drives (30:00–45:00) **5**

5 Scratch (45:00–71:00) **7**

1 Announcements and Demos (0:00–15:00)

- Check out what is possible in the programming language called Scratch that we will begin the course with!
- Be sure to check out the first-ever CS50 Puzzle Day this Saturday! Thanks to Facebook for sponsoring!
- Dinner with CS50 and Facebook is tonight at 6:30 at Maxwell Dworkin! Well, not *at* Maxwell Dworkin, but we'll meet there. RSVP here.
- As you make your way through the semester, try to keep the following snippet in mind:

  > what ultimately matters in this course is not so much where you end up relative to your classmates but where you, in Week 12, end up relative to yourself in Week 0

  There's no predetermined scale or grading curve in this course. Grades are assigned on an individual basis and take into account your background coming into the semester. There are three types of sections: for those less comfortable, those more comfortable, and those somewhere in between. Keep that in mind as you section next week!
- The first problem set was released today and is due next Friday. There will be a Walkthrough on Sunday to help get you started!
- Office Hours will start this Monday and will take place Mondays through Thursdays, 9 p.m. to midnight in the dining halls. Check out the course website for locations.
- CS50 is an amazing, empowering, exciting, liberating experience! But don't take it from me, take it from former students, teaching fellows, and even our beloved President.[1]
- Fresh off your CS50 experience, consider participating in Hack Harvard during J-Term. There you'll get the manpower you need to turn your Harvard-related final project into a full-fledged campus app.
- The Harvard College Innovation Challenge will give you a chance to receive funding for your entrepreneurial idea.
- Soon you'll be able to take advantage of the Harvard Innovation Lab which will provide you with a workspace and central hub for collaboration at Harvard.
- If you're interested in checking out what's going on in technology on the west coast, apply after September 15 for the Winter Break Silicon Valley Trip to take place during J-Term.

[1]Barack was at the CS50 Fair earlier, but he ate too much popcorn and had to leave before we could get him on tape.

- If you just want to bounce ideas off like-minded classmates, sign up for the Harvard College Entrepreneurship Forum.

2 Computer Science in the Real World (15:00–25:00)

- Keep in mind that programming is not an end in itself, but rather a means to an end. Neither is programming everything there is to computer science. Programming is simply a tool to accomplish everyday tasks much more efficiently than a human could alone.
- The real world is full of people making poor decisions related to technology. Take Firesheep as a case in point. This simple Firefox extension allows one to steal the session of a nearby user and thereby login to one or more of his online accounts over a shared, unencrypted wireless connection. By default, computers listen only for traffic that is destined for themselves, yet they are fully capable of listening to traffic destined for any computer within their range. Many websites these days continue to use a protocol that doesn't require this traffic to be secured against being intercepted. More on this later in the course!
- Another security flaw that should hit home with many of you is that of the iPhone Tracker. If you read the fine print of your iTunes agreement, you'll see that by using iTunes and your iPhone, you actually agree to have information about your location stored in iTunes and sent to Apple. Of course, there's a compelling reason for this from a usability standpoint (e.g. navigation, recommendations), but it became a huge cause for concern when this information was discovered to be unencrypted on many Apple devices. David ran this iPhone Tracker app on his own iPhone and was shocked at how granular the location information really was.
- If nothing else, after taking this course you'll have the knowledge you need to make more informed decisions related to privacy and security issues of technology.

3 From Last Time (25:00–30:00)

- Recall our peanut butter and jelly sandwich algorithm from Wednesday:

  1. Take two slices of bread from the loaf.
  2. Apply two tablespoons of jelly on both halves.
  3. Apply two tablespoons of peanut butter in the same way.
  4. Join the two halves.
  5. Have fun!

  How can we improve on this algorithm?

- Consider step 1. We forgot to mention that we need to open the bag first. But now think about a second round of sandwich making. Now the bag is already open, so we don't need a step to open it. To handle this case, let's add a condition that we only open the bag if it's closed to begin with. We might represent this in "pseudocode," that is, a mock programming language we use just to express our ideas, like so:

  ```
  if bag is closed
      open bag
  ```

  The indentation indicates that the second line of code should only be executed if the first line is true.

- Now we need to account for the loaf of bread being sliced or not. We can do this with a two-pronged condition, i.e. if-else:

  ```
  if bag is closed
      open bag
  if loaf is already sliced
      remove two slices from open end
  else if it isn't sliced
      remove bread from bag
      cut two slices of bread from loaf
  ```

  Of course, at any point, we could deteriorate to complete ridiculousness. For example, we could insist on cutting irregularly shaped slices. However, for the sake of sanity, let's keep moving forward.

- Now we place the slices of bread on the table and start thinking about the peanut butter. Here, too, we must account for the jar being either open or closed. Even if it's open, we need to account for the so-called corner case of this being the first time we *ever* opened the jar given that there is a plastic inner seal. By corner case, we simply mean a situation that we rarely have to handle but which, if left unhandled, causes unexpected behavior. So to continue our algorithm:

  ```
  if bag is closed
      open bag
  if loaf is already sliced
      remove two slices from open end
  else if it isn't sliced
      remove bread from bag
      cut two slices of bread from loaf
  put slices on table separately
  open the jar of peanut butter
  if jar is sealed
      unseal jar
  ```

```
pick up knife
put knife in peanut butter
scoop out 2 tablespoons of peanut butter
```

And so on. The takeaway here is that there are countless opportunities to make mistakes such as failing to handle corner cases.

4 Binary and Hard Drives (30:00–45:00)

- So far we've used only pseudocode, a non-programming language, to express our ideas. As we make the transition to true programming languages like C, we'll need to be extra careful about syntax. Even a single missing semicolon in millions of lines of code will break our program! Fortunately (or unfortunately?), as the semester progresses, you'll find yourself making fewer syntax errors and more interesting errors.

- Let's take a look at our very first program in C:

```
#include <stdio.h>

int
main(void)
{
    printf("hello, world\n");
}
```

  Ta-da! This program does nothing more than print "hello, world" to the screen. It is a common convention in computer science to write a so-called "hello, world" program when you are first beginning to learn a new language.

- As cryptic as this program might seem, it's still readable by a human. However, the truth is that it's not *actually* readable by a machine in its current state. Computers only really understand binary, the language of 0's and 1's. This is because 0 and 1 map nicely to the off and on states of electricity. In order to run our very first program in C, we'll need to run it through a *compiler*, a program that will translate it into binary.

- Let's begin learning binary the same way we learned decimal: counting. 0 and 1 are easy, but how do we count to 2 in binary? We've run out of digits. Recall from your days of arithmetic the concept of "carrying the 1." If we carry the 1 here, we end up with 10 as 2 and 11 as 3. Here are the numbers 0 through 4 in pretty-printed format:

```
000
001
010
```

```
011
100
```

Counting up, it seems, is relatively easy. We can have as many leading zeroes as we want when representing numbers on paper, but in practice, computers only handle a certain number of total zeroes, or bits, at a time. This is where the terms 32-bit and 64-bit come from, but more on that later.

- Given a binary number like 111, how do we convert it to decimal? Consider a number like 123 in decimal. The 1 is in the 100's column, the 2 is in the 10's column, and the 3 is in the 1's column. $123 = 1*100+2*10+3*1$. Likewise in binary, the columns represent powers of the base number. Decimal is base 10, so each column is a power of 10. Binary is base 2, so each column is a power of 2. To convert 111 to decimal, then, we take each column and multiply by the appropriate power of 2: $111 = 1*4+1*2+1*1 = 7$.

- Somewhere along the way, it was decided to lump bits together in order to create a more practical unit for measuring information. Thus the byte, which consists of 8 bits, was born.

- Now that you understand binary, you'll get the joke behind the Foxtrot comic in which Jason professes to have completed his phys-ed homework, which required him to do 100 pushups, having done only 4.

- In order to use binary to express alphabetic characters, we'll need some kind of mapping between numbers and letters. This is where ASCII comes in. To see the complete mapping, check out this ASCII table. However, you only really need to know that uppercase A is 65 and lowercase a is 97. This will help with conversions.

- If we allow ourselves eight columns, we can represent all the numbers from 0 to 255 (and their character mappings according to ASCII) in binary. These eight columns correspond to eight bits, which together make up a single byte. The letter A in binary is written as 01000001.

- How is this binary data stored in reality? Take a look at this video and this follow-up for a detailed explanation of how hard drives work. Here are the take-away points of these videos:

  - Hard drives consist of circular platters which spin as data is being written to and read from them. When data is to be stored on the hard drive, it passes from RAM along with software signals that designate how the data is to be stored. Certain signals control how the platters spin and others control the read-write heads. The distance between the heads and platters is less than the width of a human hair, yet the platters spin 5400 RPM or faster. Multiple platters make for greater efficiency than a single platter.

  - A read-write head in a hard drive contains a tiny electromagnet which conducts the software signals it is passed and, during writing, flips single bits on the platter either on or off by polarizing it in one direction or the opposite. During reading, the process is reversed and the electromagnet conducts signals from the bits on the platter. A single file may be stored in locations scattered widely across a platter, so a separate file keeps track of the locations of all the bits of files.

5 Scratch (45:00–71:00)

- The program we demonstrated at the beginning of lecture was written in a framework called Scratch, developed down the road at MIT. This framework was designed to allow students to ease into the world of programming by dragging and dropping puzzle pieces in order to create logical constructs such as if-else statements and loops.
- To start working with Scratch, you'll need to download the program from MIT's website.
- Once you install Scratch and open it, take note of the following layout:
  - On far left, notice the "palette" of puzzle pieces, which represent programming statements. Programs will be composed by putting puzzle pieces together in a particular order.
  - At bottom right are sprites, or characters that will carry out your instructions.
  - At top right is the stage, where the program will be carried out.
  - To the left of the stage is the scripts area, where puzzle pieces must be dragged and strung together.
- We can recreate our very simple C program in Scratch using the "say" block. `Hai1.sb` is equivalent to `hello.c`, only a little more colorful. The "when green flag clicked" piece is equivalent to our `main` keyword in the C program above.
- Before we go any farther, let's talk about some computer science jargon: a *statement* is an action that we give to the computer to perform. In the context of Scratch, statements begin with verbs like "say," "play," and "wait."
- Obviously, we're taking baby steps, but realize that that's what programming is all about–taking very basic building blocks and creating functions and more complicated programs.
- `Hai2.sb` is slightly more complicated. The cat will say "O hai, world!" for 1 second, wait 1 second, say it again for 1 second, wait 1 second, and say it again for 1 second.

- So far we've only made use of *statements*, which are direct imperatives given to the computer. But if we want to introduce logic into our program, we'll need *boolean expressions* and conditions. Boolean expressions are those that have only two possible values: true or false, yes or no, on or off, 1 or 0. No matter how you say it, it's a simple variable. In Scratch, boolean expressions are represented as hexagons and are written as yes-or-no questions such as "touching mouse-pointer?," "mouse down?" or comparisons such as less-than, equal-to, or greater-than.

- Boolean expressions aren't new to you. Consider on HarvardCourses when you search for courses offered this year. Somewhere in the code, that translates to a boolean variable that signifies either offered this year or not offered this year.

- `Hai4.sb` and `Hai5.sb` make use of conditions and boolean expressions. In the first, the condition $1 < 2$ always evaluates to true, so the cat meows everytime we click the green flag. In the second, however, the condition says, "pick a random number between 1 and 10 and if that number is less than 6, have the cat meow." This is what we call a pseudorandom number generator. The effect of this pseudorandom number generator on our program is that the cat will meow approximately half the time we click the green flag.

- If we want our cat to meow multiple times, we can certainly just duplicate the statements however many times we want. But this would be very tedious to drag the puzzles pieces over and over again. Instead, we can use loops when we wish to repeat a statement. In `Hai6.sb`, we implement a loop which causes the cat to meow indefinitely. Infinite loops aren't necessarily bad: consider the case of a spellchecker in a word-processing program that constantly checks for words you have typed.

- In `Hai7.sb`, we combine a loop and a condition so that the cat will meow only if the mouse pointer is touching it or, in other words, if we are petting it. In `Hai8.sb`, we add an extra condition so that the cat will meow indefinitely, but will roar if we touch it with the mouse pointer.

- *Variables* are another useful programming construct. They allow us to store information about the *state* of a program. For example, in the case of the peanut butter and jelly algorithm, we might have defined a variable `slices_of_bread_left`.

- *Arrays* are essentially collections of related variables. In the game `FruitcraftRPG.sb`, for example, an array is used to store the different types of fruit which have been collected. An array is one of many different types of data structures which are essentially buckets in which we can store information of interest to our program.

- Now is a good time to introduce the concept of *threads*. This is a fairly complicated concept which doesn't usually get introduced in the first week of a computer science course, let alone in the first semester of computer science training. However, let's Scratch the surface of a threading discussion.[2] In the context of your computer, threads are what allow you to have multiple software applications open at once. Your computer isn't actually doing multiple things at once; rather, it's switching back and forth between tasks faster than you can perceive.

- Threading refers to the notion of multiple threads of code executing simultaneously. In `Move1.sb`, we have a simple animation of a duck moving back and forth across the screen, screaming and turning around every time it touches the edge. This is a single thread. In `Move2.sb`, we achieve "multithreading," at least in appearance. In terms of programming, we have two different scripts associated with two different sprites, a cat and a bird. For the cat, we begin by placing him in a given spot on the stage and orienting him in a random direction. Then we begin a loop whereby if he touches the bird, then the game ends; otherwise, orient toward the bird and advance one step. For the bird, we again place and orient him and then move him around the stage three steps at a time. Effectively, then, the cat is chasing the bird until he catches him.

- *Events* are another method of communicating between sprites. `Marco.sb` leverages events to play the game of Marco Polo. Thus far, our sprites haven't really been communicating with each other. But in this game of Marco Polo, one sprite is saying "Marco," and the other is listening for him to say it so that she can say "Polo" in response. The second sprite is listening for the event which the first sprite broadcasts.

- Scratch also offers sensor boards which take user input in the form of sound, light, and movement, as required by the Hacker Edition of Problem Set 0.

- Check out Scratch Scratch Revolution and Raining Men for more examples of what you can do with Scratch!

[2]Yes, yes I did just make that joke.