Week 1 Monday | Computer Science 50
September 22, 2008 | Fall 2008
Scribe Notes | Anjuli Kannan

Introduction (0:00-5:30)

- In the interest of spreading geek culture, a here's an internet forward to amuse you: http://www.youtube.com/watch?v=MunMCO3uNdA
- Section this week on cs50.net

What Is a Program? (5:30-15:00)

- What we want to take away from Scratch are the programming concepts
- It will take us longer to learn to deploy these same concepts in C, PHP, and Javascipt
- So, what is a program?
  - Set of instructions that tells the computer to do something
  - An algorithm or procedure that teaches the computer how to do something
  - Serves some purpose, e.g., does mathematics
- How did it get written?
  - Someone sat down at a computer that already had years worth of software (an operating system, etc.) on it and then wrote on top of that software
  - This can be done in as simple a program as Notepad
  - You simply need to "speak the language"—in our case, C
  - But if you just save this and double click it, will it run?
  - No, it will open up Notepad again and show the *source code*, or the text instructions you wrote out
  - To run your program, you must *compile* your source code using a *compiler*
  - A compiler takes the English like syntax in a language we understand (such as C) and translates it into zeroes and ones that the computer understands (Slide 2)
- A CPU understands a very basic set of instructions—arithmetic and logic (e.g., branching, jumping)
- Back in the day, programmers would communicate in these instructions to the CPU using things like punchcards
- Today, we can work at a much higher level because people have already done the low level stuff for us
- So now when we write a program in C, we are using functions that other people have written
- For instance, printf, a function that prints to the screen, was implemented by someone else a long time ago. They wrote out the machine instructions that should be called when you simply write printf.
- Finally, after your code is compiled, you can an executable (a .exe file) that is written in zeroes and ones so the CPU can understand it
- Notice that you can't run the same programs on PCs and Macs.
- That's because PCs and Macs don't understand the same instructions, so the programs are compiled differently

- When you compile a program on a PC for a PC, it will only run on a PC
- There are also things that are platform independent such as Java

Cloud Computing (15:00-29:30)

- In this course, we'll be using Linux
- Unlike in Windows, where you click and drag a lot, in Linux, you usually type commands on a command line
- Harvard has a bunch of Linux servers called NICE (new instructional computer environment) and accessed at nice.fas.harvard.edu
- In this class, we used to do all of our work on NICE. We'd connect to nice.harvard.edu and be assigned to a particular computer.
- This year we will take advantage of *cloud computing*
- Cloud computing is the use of computers that are not physically located on your even desk or even your building
- Amazon has Elastic Compute Cloud, which we're going to use in this course for cs50.net
- You're going to connect to the cloud, which means you'll be working on some server on the east coast, but you don't know which or where
- Our server will channel you to the one that has the most computing power available
- For problem sets that require a lot of computing power, we can fire up additional computing resources
- Cloud computing is good because you might want a certain amount of computing resources most of the time and a different amount some of the time
    - E.g. if you are hosting a commercial website that occasionally gets a lot of traffic
- When we connect to a computer in the "cloud" it's actually a virtual machine
- This means that someone has written a program to simulate a CPU, and that's what you are working on
- Why is this useful? Because they can have many of these virtual machines run on a single computer
- A front end server talks to "the internet" (which is where you come from) as well as all the servers in the cloud. As requests for computing space come in, it dispatches them to an available server.
- These virtual machines can be managed by executing commands alone.
- This is really cost-effective. A virtual machine (2 gigs ram, 160 gigs storage) is 10 cents/hour.
- We can fire up resources on demand and then get rid of them.

SSH, the Unix Shell, and Nano (29:30-39:00)

- For now, we'll use nice. So how do we connect to one of these machines?
- Use an SSH (secure shell) program

- We'll start with putty. (You can download this with a simple Google search)
- When we open putty, we get a shell that asks for a user name and password
- Once we have logged in, we are connected to one of the computers in NICE in the Science Center basement
- Then we get a prompt
- Associated with your login is a directory of files. No matter which computer you're connected to, you have access to your home directory
- Some useful commands:
  - mkdir <name> – make a directory called name
  - cd <dir> - move into directory called dir
  - ls – list contents of current directory
  - mv <file1> <dest2> - move contents of file1 into file2 (rename file1 to file2)
  - pwd – print working directory (in case you forget where you are, which isn't obvious from the prompt)
  - rm <file> - remove file
- Example:
  - cp ~/public_html/lectures/weeks/1/src/* .
  - This means copy everything in the src folder into the current directory
  - cp means copy
  - * means "grab everything"
  - . means "right here"
- We'll start with a text editor called nano
- We go in by typing "nano" at the prompt and pressing enter
- Nano shows us all the commands we need to know at the bottom of the screen
- ^ means control
- So we can type some stuff and then save by pressing Ctrl+O
- We type the name "foo" and press enter
- Once we've exited, we ls again and find foo in our current directory
- To open foo, we type "nano foo" and press enter
- Now let's type in the program we've seen a few times on the slides (Slide 2)
- We type this word for word into nano, press Ctrl+X to exit, Y to save, and type in hai.c
- Now at our prompt we try typing hai.c and pressing enter
- The shell tells us, "Permission denied"
- Why? We haven't compiled it, so we can't run it

Compilation (39:00-48:00)

- We use a compiler called gcc by typing "gcc hai.c" and pressing enter at the command prompt
- This results in an executable called a.out being put in the current directory.
- Now we type a.out and press enter.

- It works: the words "hai, world!" are typed in the shell. However, its strung together in the same line as the prompt, which is a little ugly.
- We can correct this by adding "\n" to the end of the string of text. \n is an escape sequence meaning newline character.
- Once saving this and recompiling, we get the desired result.
- OK, now we have the basics. Let's see what we can do with this.
- First, can we give it a different name? We do this by typing at the command line "gcc –o hai.exe hai.c". Now we have a file called hai.exe. When we type hai.exe at the command line our program runs.
- But .exe is really a windows convention. We don't need to give our executables extensions like that.
- We rename our executable by typing "mv hai.exe hai" at the command line
- Now we can just run hai
- Later on, as these commands become more complicated, we'll use make files, which the problem set will walk you through
- We will also see some other text editors: vim and emacs

Printing Text to the Console (48:00-56:30)

- Now let's see what's going on in our program. See handout.
- There is a whole bunch more stuff on this version of the program than there was on the slide. These are comments
- Anything between /* and */ is a comment, which means it will be ignored by the compiler
- The stuff at the top just tells the reader what the program is called, who it's by, etc.
- printf is a function
- A function is like a mini program that someone else wrote that you can invoke to do some job for you
- That way, don't have to rewrite the instructions for common jobs over and over again
- In this case, printf prints text to the console
- Following printf are two parentheses with text in quotes inside them
- This text is the parameter of the function. In this case, it tells the computer what you want to be printed. In general functions can take zero or more parameters.
- If you forget how a function works (e.g. what arguments it takes) you can look up function documentation on cppreference.com by going to standard C library
- Escape sequences are sequences of characters that can go within the quotes (for instance, in the argument to printf) but will *not* be printed as they appear
    - \t tab
    - \n newline
- At the top of hai1.c we have the line "#include <stdio.h>". This tells the compiler where to find the declaration of the function printf. The standard library is built in, so the compiler will know where to find it.

- Now let's take a look at math1.c. Here, we don't need to include stdio.h because we're not printing.
- This program saves the values 1 and 2 in variables, then adds them up. But when we run this program, we have no evidence that it did anything. Nothing is printed to the console.
- You can print variables to the console using printf, but you must use format strings to tell printf what the variable contains. For instance, %d is the format string for an integer. (You can look the rest up on your own)
- We use a format string in the following way: printf("%d", z)
- That means to print the value of z. You can't simply say printf("z"), which would print a z, or printf(z), which wouldn't give printf any information about what z is. You must use the format string so that printf knows what it is printing
- We probably also want to throw a newline on the end of that: printf("%d\n", z)

Printing Variables to the Console (56:30-72:00)

- We can all the common arithmetic operations: +, -, *, /, %
- % is modulo (remainder) – e.g. 6 % 4 = 2
- Remember order of operations? Precedence in C is determined according to some similar established rules (slide 17)
- Sometimes we're not using nice neat integers. Often we're dealing with numbers that have infinitely many digits, which is problematic for computers who can only store so many digits for each number.
- A 32 bit computer can store a number of up to 32 bits, which means it can store up to 2^{32} , or ~ 4 billion
- Floating point numbers, or floats, are used to store numbers with decimal points.
- See math3.c. Notice we have to declare the type (float) of the variable answer.
- In the format string for a float, we can tell the computer how many digits we want displayed: "%.2f\n"
- But when we run this, we get that answer is 1.0
- Why? Because when we divide an integer by an integer, we get an integer and the remainder is discarded. 17 divided by 13 is 1. This value is then converted to a float, but it's too late because we've already lost the remainder.
- To fix this problem, we can do 17.0/13 (or 17/13.0 or 17.0/13.0) or use casting (see math4.c and math5.c)
- Other data types: char (character), double (64 bit floating point number)
- You can also prepend ints with things like long, short, which changes the number of bits they take up

The CS50 Library (72:00-78:30)

- Not built into C, but implemented for you in the cs50 library, are bools (Boolean expressions) and strings (sequences of characters)

- Also in the cs50 library are fuctions such as GetInt() and GetString() to fetch user input
- You can see an example of this in hai3.c, in which we get the user's name and then print it back up
- Notice that we have to include cs50.h in order to make use of GetString()
- We also must include the flag –lcs50 when compiling