**Announcements** (0:00 – 13:00)

- Remember to section
- You can now use the course bulletin board on the website. All posts are anonymous. This is a good way to get a common question answered once for all students, so please make use of it.
- Check out the bulletin board for Dr. Malan's final project suggestions. Also keep an eye out for problems on campus that could be solved by a good final project.

**Secure File Transfer** (13:00-17:00)

- SFTP is a protocol to upload files from your home computer to your FAS account
- First, download and open SecureFX (for Windows)
- Connect to nice.fas.harvard.edu or fas.harvard.edu. It doesn't matter which one because it all goes to the same place. (Note that it DOES make a difference when you SSH.)
- You'll be prompted for your FAS username and password
- When it opens you will see a window showing a graphical representation of your FAS home directory
- Drag files from your desktop or computer directory into appropriate folder in your ~/cs50/ directory
- You will be prompted to tell if ASCII or Binary
- For pure text (.txt, .c) choose ASCII
- For anything else choose binary (including .sb)

**Our First Program** (17:00-26:30)

- Here's the program we wrote and compiled last time (hello1.c)

```
#include <stdio.h>

int
main(int argc, char * argv[])
{
    printf("hello, world\n");
}
```

- Let's look at the various components in detail.
- `int main (int argc, char * argv[])`
    - This is a function. A function is like mini-program that can be called by your program in order to execute its contents.
    - Whenever a C program is run, the computer will automatically look for the main method to execute. From their, other functions may be called, but it always looks for main first.
    - If you don't have a main method, it won't compile.
- `int printf(const char *format, …);`

    - Here's another function, or a miniature program. This particular function prints text to the screen.
    - This time, we're not writing it, we're calling it.
    - When we use a function like printf(), we are making use of some code that someone else has already written for us.
    - Someone else wrote the code that actually tells the computer to print characters to the screen, and added it to the UNIX system.
    - Now rather than "reinvent the wheel," we can just call their function
    - Notice we are passing the function arguments within parentheses
    - Arguments allow you to control the behavior of the function
    - In this case, we're telling it what to print
- `#include <stdio.h>`
    - This tells the compiler we will be using some readymade functions from the stdio library (namely, printf)
    - You must include statements like this at the top of your program so the compiler knows where to find the necessary code
    - You can also write your own libraries of code and tell a program to include them. That way, you only have to write functions once and then can reuse them in many different programs.
- `"hello, world!\n"`
    - Notice the sequence \n. We can use the following combinations of characters to indicate elements of text that there is not a key on our keyboard for:
        - \n Newline
        - \r Carriage Return
        - \t Tab
    - What's the difference between a newline and a carriage return? A carriage return moves the cursor all the way to the left, but does not necessarily scroll down a line.
    - Notice also that the desired text is enclosed with double quotes. What if we want to actually display a "? What if we actually want to display a /? We use these escape sequences so as not to confuse the computer:
        - \"
        - \\

**Variable Types** (26:30-34:00)

- What can we do with C that we couldn't do with Scratch?
- The printf function can take variable input
- But when we give it variable input, we have to tell it what *type* of variable to expect
- Scratch had only one type of variable (number), but C has four types, and when we make a variable, we have to tell the computer what type of value is going to be stored in it
- char (character) – 8 bits, so you can only have 256 different characters (ASCII)

- int (integer) – 32 bits, so you can represent roughly 4 billion total numbers
  - signed int – used to represent both negative and positive numbers, so you can only go up to about 2 billion
  - unsigned int – used only to represent positive numbers, so can be as large as 4 billion
  - long – 32 bits
  - long long – 64 bits
- float (floating point value, real number) – 32 bits
- double (also real number) – 64 bits, so can be used to store larger or more precise numbers

**Format Strings** (34:00-40:00)

- When you want to print the value of a variable, you must use a placeholder in the string that corresponds to the type of the variable:
  - %c char
  - %d integer
  - %e scientific notation
  - %E scientific notation
  - %f double or float
  - %s string
  - %u unsigned int
  - %x hex
- Example: math2.c – prints out the value of z, 3

```
int
main(int argc, char * argv[])
{
    int x = 1;
    int y = 2;
    int z = x + y;
    printf("%d", z);
}
```

- The sizeof() function tells us how many bytes are taken up by a variable
- Example: sizeof.c – prints out the number of bytes in each variable type

```
int
main(int argc, char * argv[])
{
    /* local variables */
    char c;
    double d;
    float f;
    int i;

    /* report the sizes of variables' types */
    printf("char: %d\n", sizeof(c));
    printf("double: %d\n", sizeof(d));
```

```
    printf("float: %d\n", sizeof(f));
    printf("int: %d\n", sizeof(i));
}
```

- The above outputs the following

```
char:  1
double: 8
float: 4
int: 4
```

- Notice that some of the text in sizeof.c is contained in between /* and */
- Anything between /* and */ is called a comment and is not read by the compiler
- Comments are reminders to yourself, explanations of how your code works, or instructions to your teaching fellow
- Comments are an important aspect of style because they make your code readable

**Arithmetic Operators and More Formatting** (40:00-49:00)

- C understands all of the arithmetic operators we are familiar with: +, -, /, *
- In addition, we can make use of the modulus operator %
- % tells us the remainder of a division:
  - 10 % 2 = 0
  - 11 % 2 = 1
- Note that these operators have precedence rules which can be referenced in a C manual or online
- What if we execute this code? `float answer = 17 / 13;`
- `answer` will take the value 1 because 17 and 13 are integers and integer division discards remainders
- In order to get an exact answer, we must make 13 or 17 a float by doing any of the following
  - `float answer = 17 / 13.;`
  - `float answer = 17 / 13.0;`
  - `float answer = 17 / (float) 13;`
  - `float answer = 17.0 / 13;`
- We can also indicate the width and precision to be used when printing variables in the following format: `%<width>.<precision>f`
- Width – how many spaces the number should take up total (including both the number itself and white space on the left)
- Precision – how many digits to include
- Example: math3.c

```
int
main(int argc, char * argv[])
{
    float answer = 17 / 13.;
```

```
        printf("%.2f\n", answer);
    }
```

- The above will output 1.3
- C is missing two useful variable types: Booleans (true/false) and strings
- We have written a cs50 library that includes these two
- Example: hello2.c

```
#include <cs50.h>
#include <stdio.h>

int
main(int argc, char * argv[])
{
    string name = "David";
    printf("hello, %s\n", name);
}
```