CS50 Section
TF: Sam Oh
Email: soh02@college.harvard.edu
Week 2

Expectations for Class

1. Please watch _____ before-hand, we will usually spend the last part of class going through questions about lectures, the pset spec, etc.

List of Resources

1. Office Hours (HSA and Large)
2. Sections! (and TF)
3. Walkthroughs
4. Lecture Notes
5. Shorts
6. Study50

By the end of class you should be comfortable with

1. Arrays
2. Functions
3. Command Line Arguments
4. Directives
5. Scope

Arrays

Arrays are **data structures** that allow us to store data of the same type in memory locations.

To better understand the concept of an array, think back to the last time you picked up mail in the Science Center basement or your house's mailroom.

An array is simply a block of space in memory (the mail center) that has been partitioned into identically-sized chunks (mailboxes). Each chunk can store a certain amount of data (mail) that can be accessed by an index number (mailbox number).

An array is declared with the following syntax: int student_grades[12];

This would create an array of 12 integers. The array name is student grades. I can access individual elements of the array easily: student_grades[5] = 98; student_grades[10] = 85;

Arrays can also be multidimensional, and individual elements can be accessed in an identical manner: char tic_tac_toe[3][3]; tic_tac_toe[1][2] = 'X'; double hypercube[20][20][20][20]; hypercube[4][9][0][15] = 6.2569;

String my_professor = "David Malan"

1. What char should my_professor[0] return?

2. What array index will return "M"?

3. What would happen if you returned my_professor[11] or my_professor[12]?

Functions

Function declarations have three parts: a **type**, a **name**, and a comma-separated, **argument list**, each argument having a type and a name. They end with a semicolon. The following are examples of valid function declarations: **(add your own below!)**

int add_two_nums(int arg1, int arg2);

char first_letter_is(string word);

The following are examples of invalid function declarations. Why are each of these invalid?

int (char letter1, char letter2); // no name

simple_function(int input); // no type

int higher_number(int num1; int num2); // argument list not comma-separated

A function definition should occur **after** and **separately** from the function declaration. (This rule is not hard and fast, but it is good practice.) The beginning of the function definition should match, perfectly, the function declaration (except for the semicolon). As an example, going off of the add two nums() function declared above:

int add_two_nums(int arg1, int arg2)

```
{
        int sum = 0;
        sum = arg1 + arg2;
        return sum;
}
```

Command Line Arguments

argc is an integer variable, provided by **main()** that tells you how many command-line arguments were inputted. **argv** is an array of **strings (or, char *s)**, that contain the actual command-line arguments themselves. You can manipulate both, e.g.:

./this class is cool

1. What is argc?
2. What would argv[0] be?
3. How could you print cool?

Directives

The directives (or, macros) that you encounter most are **#include** and **#define**. When the compiler sees **#include**, it essentially copies the contents of the file you list into your program's object code.

When the compiler sees **#define**, it goes through and substitutes any instances of the "symbol" you use to represent the "magic number" (which could also be a letter, a word, or even a small function!) with that "magic number".

Can anyone explain what a "magic number" is and why it isn't good to use in a program?

```
#include <cs50.h>
#define YEAR 2013
```

The former would paste the entire contents of cs50.h **atop the .c file** that contains the #include. The latter would literally **replace all instances of** YEAR with "2013" in your object code. Make sure NOT to put semicolons at the end of your #defines!

Scope

A variable's scope is a characteristic of how visible that variable is to other functions. A variable's scope can be global, or local. In the below example, a is global, and can be used and manipulated by all functions. b is local to main() and c is local to f1(). d is local only within the context of the for loop in main(). That is, d means nothing anywhere else but while the program is running through that loop.

```
void f1();

char a = 'i';

int main(void)
{
        int b = 10;
        for (int d = 1; d <= b; d++) // loops with only one statement don't need braces
                f1();
        return 0;
}
```

```
void f1()
{
        char c = ‘h’;
        printf("%c%c\n", a, c);
}
```

Questions, Problem Set 2, etc