CS354: Lec 001

Grade Distributions

- Projects: 55%
- Quizzes: 10%
- Midterm: 15%
- Final: 20%
- Participation: Irrelevant.

There will be 1 quiz per week, similar to CS252.

Letter grades are as follows:

- 95%: A
- 90%: AB
- 85%: B
- 80%: BC
- 70%: C
- 60%: D

Exams are open-note; you have the Internet to help you.

Projects should be started when they are handed out.

All projects are due at 11:59 P.M., but it's more like whenever they start grading (e.g. next morning). Work less than a week late will receive half credit, and will not be accepted after said week.

How Computers Work

- Computers are built with three main parts
  - *Apps*: Word Processing, Browsers, etc.
  - *Software Systems*: OS, Drivers
  - *Hardware*: CPU/Memory/Disk
- When we run a program, such as this one:

```c
// sum.c
...
a = 1;
b = 2;
c = a + b;
...
```

- The Von Neumann Architecture, made in 1950, states that a computer works something like this:

$$\textbf{Inputs} \implies \textbf{CPU} \iff \textbf{Memory} \implies \textbf{Outputs}$$

- When `sum.c` is run, it is stored in the disk, where we use a compiler (GCC in this case) to convert the instructions to binary. This binary representation is called an object file, or machine code.
- This binary code is also stored in the disk.
- When we actually run the code (`./sum`), the OS will run a micro-program called a *loader*, which takes the machine code and stores it (and the data, in this case a=1, b=2, c=0) onto the memory.
- The processor has little memory stores too, called *registers*, which takes the required data from the memory and performs operations on it. To perform `c = a + b`, it needs the values for `a` and `b`, which it stores in the first two registers, respectively. In the third register, it can calculate the value of `c` by adding the first two registers together. Then, it loads the value from the register back into the memory; this is called a *store* operation.
- A CPU's processing can be broken down into three parts:
  - *FETCH*: Get the instruction from the memory; in this case, it's taking `a` and `b` from the memory
  - *DECODE*: Figuring out what instruction it has to do; in this case, it has to add `a` and `b` and load `c` back into the memory
  - *EXECUTE*: Executing the instruction
  - This set of actions repeats in a loop until the program is completed.
- This is what happens when the code is run, but what happens when our code is compiled first?
  - *PREPROCESSOR (CPP)*: Removes the comments, all the stuff that the compiler doesn't really need (`-E`)

- *COMPILER PROPER (CC1)*: Produces code that is written in Assembly, still human-readable, but a bit harder (`-S`)
- *ASSEMBLER (AS)*: Produces an object file, written in machine code (`-c`)
- *LINKER (LD)*: Combines multiple needed object files to create the *executable*

Basic C

```c
#include<stdio.h> // library to get/show input/output; for the printf function

int main() { // main function
    int a; // initializes variables
    int b;
    int c;
    a = 1; // sets variables to values
    b = 2;
    c = a + b;
    printf("The sum is %d", c); // The printf function takes a string and a list of
variables. The variables are inserted wherever a % is typed. (The %d just ensures that the
variable is represented as a decimal.)

    return 0; // kind of like a return true
}
```