Welcome to CS50 section! This is Week 4.

~~Please open your CS50 IDE and run this in your console:~~

~~`cd ~/workspace/cs50-section` ↵~~

~~`git reset --hard` ↵~~

~~`git pull`~~

If new to this section, visiting, or want to “start over”, run this in your console:

```
rm -r -f ~/workspace/cs50-section/ ↵
cd ~/workspace ↵
git clone https://github.com/bw/cs50-section.git
```

Start early on pset 4, it’s tough!

Know before attempting pset 4:

- Redirection methods
- Writing to/reading from file

- Memory management
  - Heap and stack
  - Dynamically allocated memory

- Pointers

- Hexadecimal

- Structs
  - Accessing fields in structs

Redirection

Useful for a variety of things:

- Grabbing the output of a command
- Putting something into a command

- We want the data into a file, not just shown

Redirection

Using > and | controls the input and output of a program.

>	Output to file
>>	Output and append
2>	Output only error messages
<	Input to file
|	Take the output of one, and use it as input for another

File I/O

The ability to read data from and write data to files is the primary means of storing persistent data, data that does not disappear when your program stops running.

- The abstraction of files that C provides is implemented in a data structure known as a `FILE`.
- Almost universally when working with files, we will be using pointers to them, `FILE*`

File I/O

- Find file manipulation in `stdio.h`

- Common file I/O functions
  - `fopen()`
  - `fclose()`

  - `fgetc()`
  - `fputc()`

  - `fread()`
  - `fwrite()`

File I/O

Switching to CS50-standard slides...

Good file I/O structure

```
#include <stdio.h>

int main(void) {
  // open file "input.txt" in read only mode
  FILE* in = fopen("input.txt", "r");

  // always make sure fopen() doesn't return NULL!
  if (in == NULL) return 1;

  // open file "output.txt" in write only mode
  FILE* out = fopen("output.txt", "w");

  // make sure you could open file
  if (out == NULL) return 2;
```

Good file I/O structure

```
  // get character
  int c = fgetc(in);

  while (c != EOF) {
    // write chracter to output file
    fputc(c, out);
    c = fgetc(in);
  }

  // close files to avoid memory leaks!
  fclose(in);
  fclose(out);
}
```

More slides

- Dynamic memory allocation
- Pointers

(Also will be posted to brandon.wang/cs50 after section)

Structures

Encapsulate data together.

- This is C's answer/precursor to object oriented programming
  - Smarter way of programming

Structures → Make a struct

```
struct student {
  char first_name[50];
  char last_name[50];
  char hometown_city[50];
  char hometown_state[2];
  int class_year;
}
```

Structures → Use the struct

```
struct student brandon;
```

Structures → Assign and access fields

```
struct student brandon;

brandon.first_name = “Brandon”;
brandon.last_name = “Wang”;

printf(“%s”, brandon.first_name); // Prints “Brandon”

printf(“%s”, brandon.hometown_city); // Error
```

That’s all for today!