Section 4

Agenda

- Reminder: Quiz next Wednesday!
  - Send in review topics by Friday!
- Redirection
- File I/O
- Pointers and Dynamic Memory

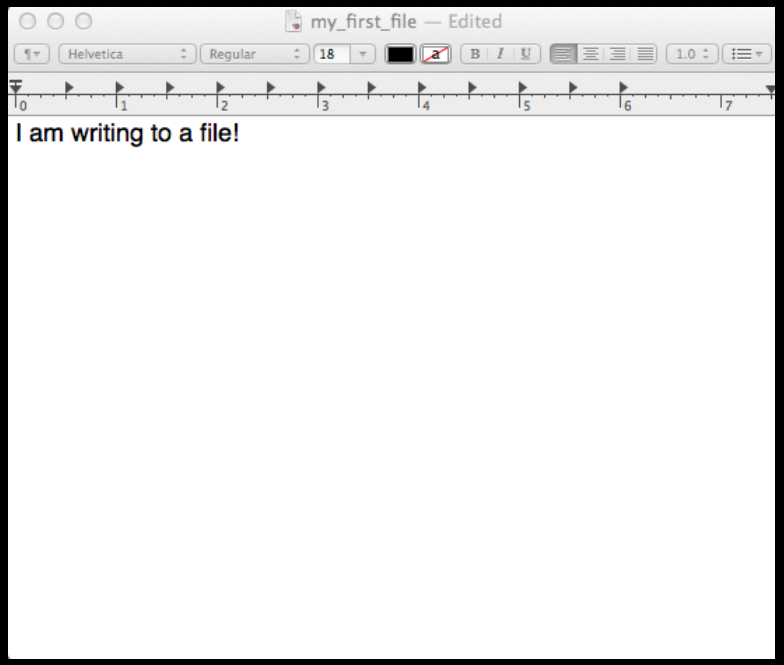
Redirection

- `>` -- output; print the output of a program to a file instead of `stdout` e.g. `./hello > output.txt`
  - `>>` -- appends to an output file instead of overwriting the data
  - `2>` -- Like above but prints only error messages
- `<` -- input; use the contents of some file as input to a program e.g. `./hello < input.txt`
- `|` -- pipe; take the output of one program and use it as input to another

File I/O

We are used to reading from and writing to the terminal:

- read from `stdin`
- write to `stdout`



But we can also read from and write to files!

Step 1: Create a reference to the file

```
FILE* file;
```

Step 2: Open the file

```
file = fopen("file.txt", "r");
```

- 1st argument -- path to the file
- 2nd argument -- mode
  - "r" -- read, "w" -- write, "a" -- append

Step 3a: Read from the file

- `fgetc` -- returns the next character
- `fgets` -- returns a line of text
- `fread` -- reads a certain # of bytes and places them into an array
- `fseek` -- moves to a certain position

Step 3b: Write to the file

- `fputc` -- write a character
- `fputs` -- returns a line of text
- `fprintf` -- print a formatted output to a file
- `fwrite` -- write an array of bytes to a file

Step 4: Close the file

```
fclose(file);
```

Remember!

- Always open a file before reading from or writing to it
- Always close a file if you open it

Example #1
Writing to a file

```c
#include <stdio.h>

#define STUDENTS 3

int main(void)
{
    int scores[] = { 96, 90, 83 };
    FILE* file = fopen("database", "w");
    if (file != NULL)
    {
        for (int i = 0; i < STUDENTS; i++)
        {
            fprintf(file, "%i\n", scores[i]);
        }
        fclose(file);
    }
}
```

Example #2

hat does this program do?

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    if (argc < 2)
    {
        printf("Usage: cat file [file ...]\n");
        return 1;
    }
    for (int i = 1; i < argc; i++)
    {
        FILE* file = fopen(argv[i], "r");
        if (file == NULL)
        {
            printf("cat: %s: No such file or directory\n", argv[i]);
            return 1;
        }
        for (int c = fgetc(file); c != EOF; c = fgetc(file))
        {
            putchar(c);
        }
        fclose(file);
    }
    return 0;
}
```

Exercise:
Writing to a file

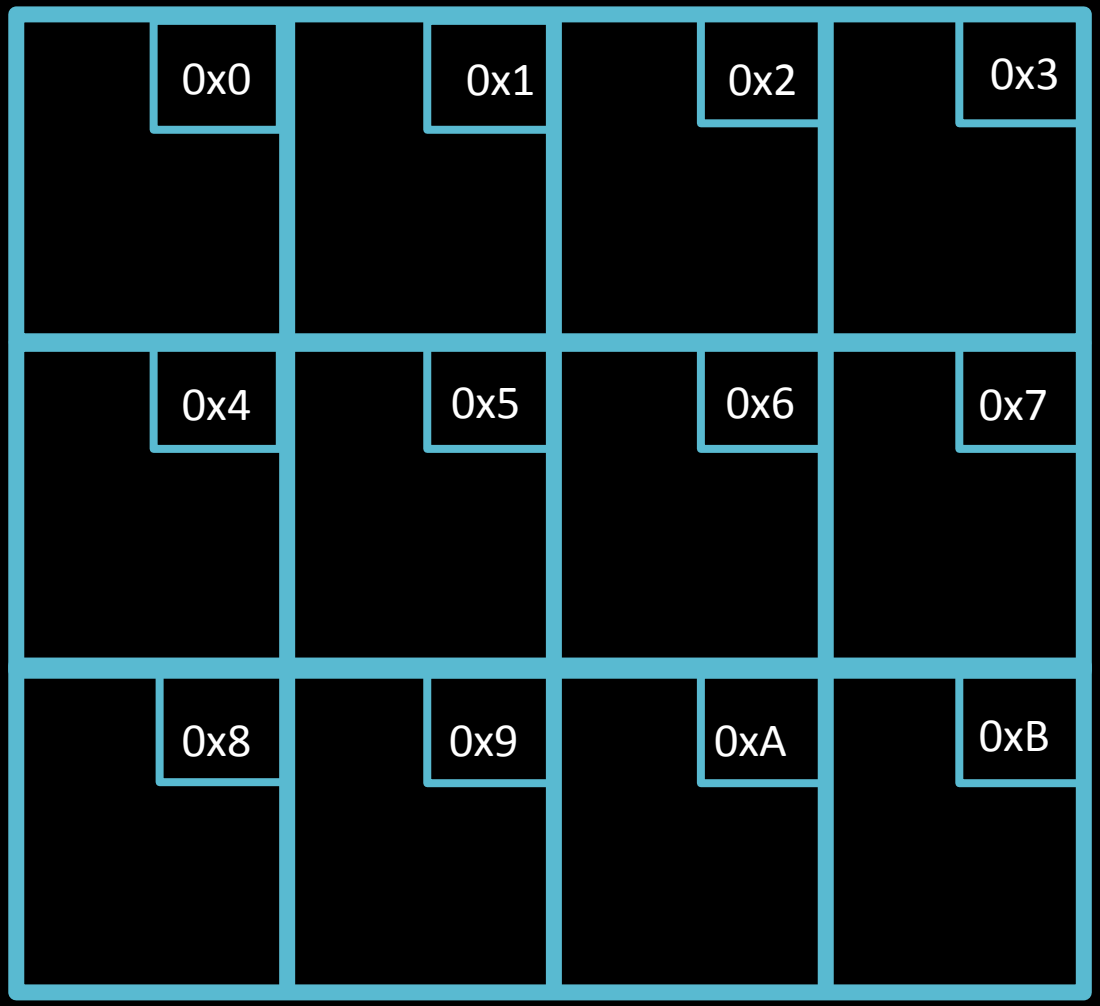
```c
#include <stdio.h>

int main(void)
{
    //insert code here!
}
```

Exercise: Writing to a file

```
#include <stdio.h>

int main(void)
{
  FILE* file = fopen(“hellp”, “w”);
    if (file != NULL)
    {
      fprintf(file, “Hello, world!”);
      fclose(file);
    }
}
```

Pointers

Memory




MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

Creating Pointers

Declaring pointers:
<type>* <variable name>

Examples:

```
int* x;
char* y;
float* z;
```

Referencing and Dereferencing

Referencing:
&<variable name>

Dereferencing:
***<pointer name>**

Under the hood...

```
int x = 5;

int* ptr = &x;

int copy = *ptr;
```

Variable	Address	Value
x	0x04	5
ptr	0x08	0x04
copy	0x0C	5

Track the values

	x	ptr
int x = 5;	5	
int* ptr = &x;	5	&x
*ptr = 35;	35	&x

Test Yourself

```
int a = 3, b = 4, c = 5;
int* pa = &a, *pb = &b, *pc = &c;
```

	a	b	c	pa	pb	pc
a = b * c;						
a *= c;						
b = *pa;						
pc = pa;						
*pb = b * c;						
c = (*pa) * (*pc);						
*pc = a * (*pb);						

Answers

```
int a = 3, b = 4, c = 5;
int* pa = &a, *pb = &b, *pc = &c;
```

	a	b	c	pa	pb	pc
a = b * c;	20	4	5	&a	&b	&c
a *= c;	100	4	5	&a	&b	&c
b = *pa;	100	100	5	&a	&b	&c
pc = pa;	100	100	5	&a	&b	&a
*pb = b * c;	100	500	5	&a	&b	&a
c = (*pa) * (*pc);	100	500	10000	&a	&b	&a
*pc = a * (*pb);	50000	500	10000	&a	&b	&a

Pointer Arithmetic

Adding/subtracting n adjusts the pointer by

`n * sizeof(<type of the pointer>)` bytes

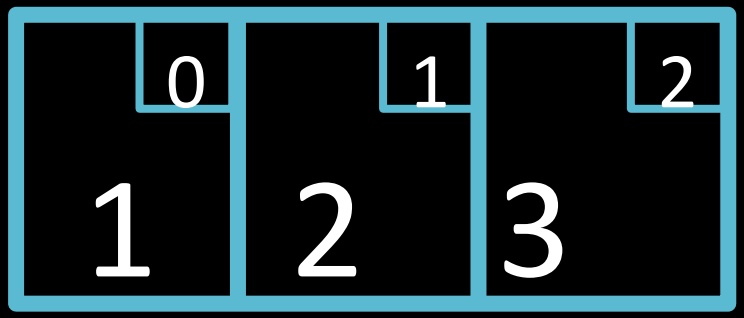
	x	y
int x = 5;	5	
int* y = &x;	5	0x04
y += 1;	5	0x08

What will print?

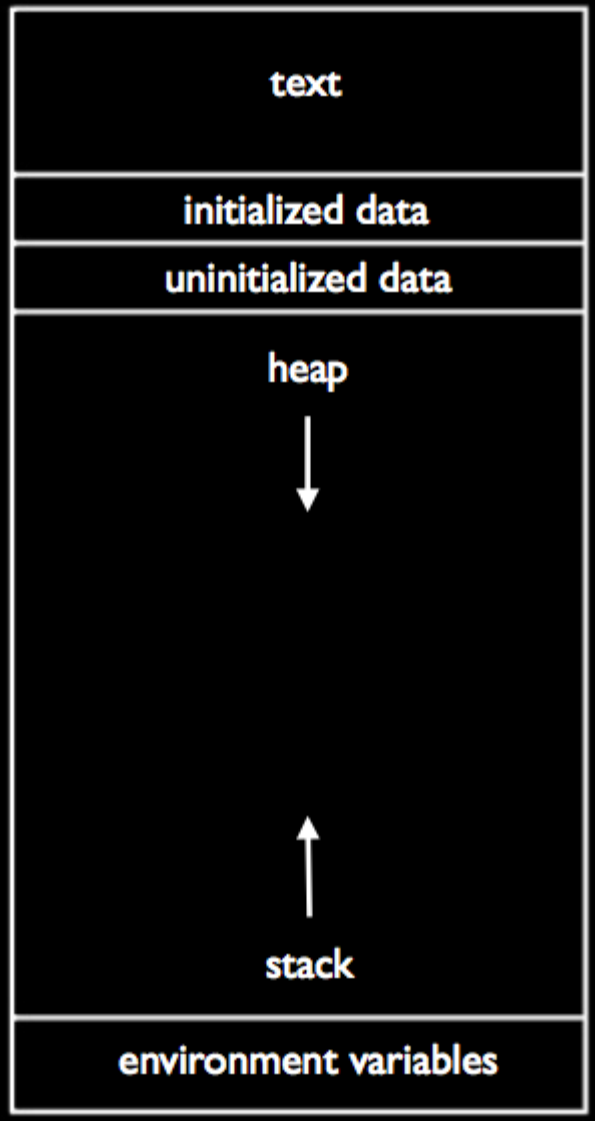
```c
int main(void)
{
    char* str = "happy cat";
    int counter = 0;

    for (char* ptr = str; *ptr != '\0'; ptr++)
    {
        counter++;
    }

    printf("%d\n", counter);
}
```

Pointers and Arrays

```
int array[3];

*array = 1;
*(array + 1) = 2;
*(array + 2) = 3;
```



Dynamic Memory Allocation



A call to `malloc()`

prototype:

```
void* malloc(size in bytes);
```

example:

```
int* ptr = malloc(sizeof(int) * 10);
```

Check for NULL!

```
int* ptr = malloc(sizeof(int) * 10);

if (ptr == NULL)
{
    printf("Error -- out of memory.\n");
    return 1;
}
```

A call to `free()`

prototype:

```
void free(pointer to heap memory);
```

example:

```
free(ptr);
```

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

int main(void)
{
    int* ptr = malloc(sizeof(int));
    if (ptr == NULL)
    {
        printf("Error -- out of memory.\n");
        return 1;
    }

    *ptr = GetInt();
    printf("You entered %d.\n", *ptr);

    free(ptr);
}
```