Week 13 - Friday

CS222
Last time

- What did we talk about last time?
- Networking practice
Questions?
Project 6
HOW TO WRITE GOOD CODE:

START PROJECT.

DO THINGS RIGHT OR DO THEM FAST?

FAST

CODE FAST

RIGHT

DOES IT WORK YET?

NO

ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.

NO, AND THE REQUIREMENTS HAVE CHANGED.

ARE YOU DONE YET?

NO

THROW IT ALL OUT AND START OVER.

WHAT?

GOOD CODE
Function Pointers
Function pointers

- C can have pointers to functions
- You can call a function if you have a pointer to it
- You can store these function pointers in arrays and structs
- They can be passed as parameters and returned as values
- Java doesn't have function pointers
  - Instead, you pass around objects that have methods you want
  - C# has delegates, which are similar to function pointers
Why didn't we cover these before?

- K&R group function pointers in with other pointers
- I put them off because:
  - They are confusing
  - The syntax to declare function pointer variables is awful
  - They are not used very often
  - They are not type-safe
- But you should still know of their existence!
Declaring a function pointer

- The syntax is a bit ugly
- Pretend like it's a prototype for a function
  - Except take the name, put a * in front, and surround that with parentheses

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

int main()
{
    double (*root)(double); // pointer named root
    root = &sqrt; // note there are no parentheses
    printf( "Root 3 is %lf", root(3) );
    printf( "Root 3 is %lf", (*root)(3) ); // also legal

    return 0;
}
```
A more complex example

- Some function's prototype:

```c
int** fizbin(char letter, double length, void* thing);
```

- Its (worthless) definition:

```c
int** fizbin(char letter, double length, void* thing)
{
    return (int**)malloc(sizeof(int*)*50);
}
```

- A compatible function pointer:

```c
int** (*pointer)(char, double, void*);
```

- Function pointer assignment:

```c
pointer = fizbin;
```
Just to be confusing, C allows two different styles for function pointer assignment and usage.

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

int main()
{
    int (*thing) (); //pointer named thing
    thing = &main;  //looks like regular pointers
    thing = main;   //short form with & omitted

    (*thing)();     //normal dereference
    thing();       //short form with * omitted

    return 0;
}
```
Why would we want function pointers?
Consider a bubble sort that sorts an array of strings

- The book uses quicksort as the example, but I don't want to get caught up in the confusing parts of quicksort

```c
void bubbleSort(char* array[], int length)
{
    char* temp;
    int i, j;
    for(i = 0; i < length - 1; i++)
    {
        for(j = 0; j < length - 1; j++)
        {
            if(strcmp(array[j], array[j+1]) > 0)
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```
Now consider a bubble sort that sorts arrays of pointers to single `int` values.

```c
void bubbleSort(int* array[], int length)
{
    int* temp;
    int i, j;
    for(i = 0; i < length - 1; i++)
        for(j = 0; j < length - 1; j++)
            if(*(array[j]) > *(array[j+1]))
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
}
```
Let's pause for a moment in our consideration of sorts and make a struct that can contain a rectangle

typedef struct 
{ 
    double x;         //x value of upper left
    double y;         //y value of upper left
    double length;
    double height;
} Rectangle;
Now consider a bubble sort that sorts arrays of pointers to Rectangle structs
- Ascending sort by x value, tie-breaking with y value

```c
void bubbleSort(Rectangle* array[], int length)
{
    Rectangle* temp;
    int i, j;
    for(i = 0; i < length - 1; i++)
        for(j = 0; j < length - 1; j++)
            if(array[j]->x > array[j+1]->x ||
                (array[j]->x == array[j+1]->x &&
                 array[j]->y > array[j+1]->y))
            {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
}
```
We can write a bubble sort (or ideally an efficient sort) that can sort anything
- We just need to provide a pointer to a comparison function

```c
void bubbleSort(void* array[], int length, int (*compare)(void*, void*))
{
    void* temp;
    int i, j;
    for( i = 0; i < length - 1; i++ )
        for(j = 0; j < length - 1; j++ )
            if(compare(array[j],array[j+1]) > 0)
                {
                    temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
}
Typechecking

- Function pointers don't give you a lot of typechecking
- You might get a warning if you store a function into an incompatible pointer type
- C won't stop you
- And then you'll be passing who knows what into who knows where and getting back unpredictable things
Simulating OOP

- C doesn't have classes or objects
- It is possible to store function pointers in a struct
- If you always pass a pointer to the struct itself into the function pointer when you call it, you can simulate object-oriented behavior
- It's clunky and messy and there's always an extra argument in every function (equivalent to the `this` pointer)
- As it turns out, Java works in a pretty similar way
  - But it hides the ugliness from you
Upcoming
Next time...

- Introduction to C++
Reminders

- Keep working on Project 6
  - It's tough!