Week 2 - Friday

CS222

Last time

- What did we talk about last time?
- Two's complement
- Floating point representation
- Math library

Questions?

Project 1

Quotes

It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so.

Mark Twain

Preprocessor Directives

Preprocessor directives

- There are preprocessor directives which are technically not part of the C language
- These are processed before the real C compiler becomes involved
- The most important of these are
  - **`#include`**
  - **`#define`**
  - Conditional compilation directives

#include

- You have already used **#include** before
  - **#include <stdio.h>**
- It can be used to include any other file
  - Use angle brackets (**< >**) for standard libraries
  - Use quotes (**" "**) for anything else
- It literally pastes the file into the document where the **#include** directive is
- Never **#include .c** files (executable code), only **.h** files (definitions and prototypes)
- It is possible to have a circular include problem

#define

- The primary way to specify constants in C is with a **#define**
- When you **#define** something, the preprocessor does a find and replace
  - Don't use a semicolon!
- **#define** directives are usually put close to the top of a file, for easy visibility

```
#define SIZE 100

int main()
{
	int array[SIZE];
	int i = 0;
	for( i = 0; i < SIZE; i++ )
		array[i] = i*i;

	return 0;
}
```

#define macros

- You can also make macros with **#define** that take arguments

```
#include <math.h>
#define TO_DEGREES(x) ((x) * 57.29578)
#define ADD(a,b) ((a) + (b))

int main()
{
	double theta = TO_DEGREES(2*M_PI);
	int value = ADD(5 * 2, 7);

	return 0;
}
```

- You need to be careful with parentheses
- Constants and macros are usually written in **ALL CAPS** to avoid confusion

Conditional compilation

- You can use directives **`#if`**, **`#ifdef`**, **`#ifndef`**, **`#else`**, **`#elif`** and **`#endif`**
- These are mostly used to avoid infinite include problems
- Sometimes they will change what gets compiled based on compiler version, system libraries, or other stuff

```
#ifndef SOMETHING_H
#define SOMETHING_H

int something(int a, int b);
#endif
```

Other C Language Features

sizeof

- We said that the size of **`int`** is compiler dependent, right?
  - How do you know what it is?
- **`sizeof`** is a built-in operator that will tell you the size of an data type or variable in bytes

```
#include <stdio.h>

int main()
{
      printf("%d", sizeof(char));
      int a = 10;
      printf("%d", sizeof(a));
      double array[100];
      printf("%d", sizeof(array));

      return 0;
}
```

const

- In Java, constants are specified with the **`final`** modifier
- In C, you can use the keyword **`const`**
- Note that **`const`** is only a suggestion
  - The compiler will give you an error if you try to assign things to **`const`** values, but there are ways you can even get around that

```
const double PI = 3.141592;
```

- Since you can dodge **`const`**, it isn't strong enough to be used for array size
- That's why **`#define`** is more prevalent

Single Character I/O

getchar()

- We haven't talked about any input in C yet
- To read the next character from input, you can use the **`getchar()`** function
- It will return the value of the next character (as an **`int`**) or **`-1`** if the end of the file is reached
  - Store the value as an **`int`** first to check to see if the end of the file has been reached
  - If not, you can then store it as a **`char`**

```
int value = getchar();
if( value == -1 )
    printf("End of file!");
```

putchar()

- **putchar()** is the output equivalent of **getchar()**
- It outputs a single character at a time
- You could use **printf()** with the **%c** formatter instead, but **putchar()** can be more convenient for single characters

```
char letter = 's';
putchar('q');
putchar(letter);
```

Lab 2

Upcoming

Next time…

- **Class is canceled Monday**
  - **As is my 2-3:20pm office hours on Monday**
  - **And my 1-3pm office hours on Tuesday**
- System limits
- Bitwise operators
- Operator precedence
- Selection

Reminders

- **Class is canceled Monday**
- Keep reading K&R chapter 2
- Keep working on Project 1
  - Due next Friday