Week 12 - Friday

COMP 2400

Last time

- What did we talk about last time?
- File systems
- Function pointers

Questions?

Project 6

Quotes

Computer Science is no more about computers than astronomy is about telescopes.

Attributed to Edsger Dijkstra
(But almost certainly not said by him)

C++

Overview

- C++ is based on C and easier to use
  - You can declare variables anywhere
    - Not the case in the C89 standard (where all variables had to be declared right after a block starts), but our `gcc` is following the C99 standard
  - It has function overloading
  - Most people think the I/O is cleaner
- The big addition is OOP through classes
- It's an approximate superset of C that includes most C structures

Compiling C++

- **`gcc`** used to stand for the GNU C Compiler
  - When it became a suite of tools used for things other than C, they changed the name to the GNU Compiler Collection
- The compiler for C++ is called **`g++`** and is part of **`gcc`**, but it may need to be installed separately
- C++ files have the extensions **`.cc`**, **`.cpp`**, **`.cxx`**, **`.c++`**, and **`.C`**
  - I prefer **`.cpp`**, but **`.cc`** is also common

```
g++ thing.cpp -o program
```

C++ is kind of an abomination

- C has too many ways to do things, but C++ is an order of magnitude worse
- Syntax is a big mess of overlapping, ambiguous ideas
  - Which only got worse in the C++11 standard and beyond, which we aren't talking about
- C++ tried to be reverse compatible with C, but not strictly true
- It tried to be object-oriented, but not strictly true
- The Standard Template Libraries are hideous compared to the Java Collection Framework
- At the time, it was the best choice available for OOP, and now we're stuck with it

Hello, World in C++

- It's not too different from C
- We need different headers for C++ I/O

```
#include <iostream>

using namespace std;

int main()
{
	cout << "Hello, world!" << endl;
	return 0;
}
```

Output in C++

- Output uses the **`cout`** object (of type **`ostream`**)
- Instead of using formatting strings, **`cout`** uses the idea of a stream, where objects are placed into the stream separated by the extraction operator **`<<`**
- The **`endl`** object adds a newline to the stream
  - Of course, **`"\n"`** works too

```
int x = 50;

cout << "There are " << x << " ways to leave your lover."
<< endl;
```

Formatting output

- Basic output is easier
- What about setting the width or precision?
- You need to include the **iomanip** header
- Put **setw(** *width* **)** in the stream to make the items take up the specified width
- Put **setprecision**(*precision*) in the stream to show a certain number of decimal places
- Put **fixed** to force it to pad with zeroes when there isn't enough precision

```
double dollars = 2.0;
cout << "Give me $" << setw(10) << fixed << setprecision(2)
<< dollars << setw(0) << "!" << endl;

// printf equivalent
printf("Give me $%10.2f!\n", dollars);
```

Input in C++

- Input uses the **`cin`** object (of type **`istream`**)
- **`cin`** also uses the idea of a stream, where items are read from the stream and separated by the insertion operator **`>>`**
- It reads items using whitespace as the separator, just like **`scanf()`**

```
int x = 0;
int y = 0;
int z = 0;
cout << "Enter the x, y, and z values: ";
cin >> x >> y >> z;
```

The `string` class

- Like Java, C++ has a class for holding strings, which makes life *much* easier
  - It's called **`string`** (with a lower case **`'s'`**)
- You must include **`<string>`** to use it
- Unlike **`String`** in Java, **`string`** is mutable
  - You can use array-style indexing to get and set individual characters

```
string a = "Can I kick it?";
string b = "Yes, you can!";
string c = a + " " + b;
c[0] = 'D';
c[1] = 'i';
c[2] = 'd';
cout << c << endl; // prints "Did I kick it?  Yes, you can!"
```

The `std` namespace

- Java uses packages to keep different classes with the same name straight
- C++ uses namespaces
- The standard library includes I/O (**<iostream>**), the string class (**<string>**), STL containers (**<vector>**, **<list>**, **<deque>**, and others)
- If you use these in your program, put the following after your includes

```
using namespace std;
```

- The alternative is to specify the namespace by putting the it followed by two colons before the class name

```
std::string name = "Ghostface Killah";
```

Functions in C++

- Regular C++ functions are very similar to functions in C
- A big difference is that prototypes are no longer optional if you want to call the function before it's defined
- Unlike C, function overloading is allowed:

```
int max(int a, int b)
{
      return a > b ? a : b;
}

int max(int a, int b, int c)
{
      return max( a, max( b, c));
}
```

Pass by reference

- In C, all functions are *pass by value*
  - If you want to change an argument, you have to pass a pointer to the value
- In C++, you can specify that a parameter is *pass by reference*
  - Changes to it are seen on the outside
  - You do this by putting an ampersand (**&**) before the variable name in the header

```
void swap(int &a, int &b)
{
      int temp = a;
      a = b;
      b = temp;
}
```

Pass by reference continued

- Pass by reference is a useful tool
- You don't have to pass nearly as many pointers
- If you want to change a pointer, you can pass it by reference instead of passing a pointer to a pointer
- It does allow more mistakes
  - Leave off the ampersand and your function does nothing
  - Change things you didn't intend to change
- You cannot pass a literal by reference

```
swap(3, 9); // Doesn't compile
```

Default parameter values

- C++ also allows you to specify default values for function parameters
- If you call a function and leave off those parameters, the default values will be used
- Default parameters are only allowed for the rightmost grouping of parameters

```
void build(int width = 2, int height = 4)
{
     cout << "We built this house with " << width
     << " by " << height << "s.";
}
```

```
build();       //We built this house with 2 by 4s.
build(3);      //We built this house with 3 by 4s.
build(6, 8);   //We built this house with 6 by 8s.
```

C++ example

- Let's write a complete C++ program that reads in:
  - A string
  - An integer
- Then, it prints out the string however many times the integer specified

More C++

The new keyword

- When you want to dynamically allocate memory in C++, you use **`new`** (instead of **`malloc()`**)
  - No cast needed, no matter the compiler
  - It "feels" a lot like Java

```
int* value = new int();          // Make a single int
int* array = new int[100];       // Array of ints
Wombat* wombat = new Wombat(); // Make a Wombat
// Make 100 Wombats with the default constructor
Wombat* zoo = new Wombat[100];
```

The delete keyword

- When you want to free dynamically allocated memory in C++, use **delete** (instead of **free()**)
  - If an array was allocated, you have to use **delete[]**

```
int* value = new int();
delete value;

Wombat* wombat = new Wombat();
delete wombat;

Wombat* zoo = new Wombat[100];
delete[] zoo; // Array delete needed
```

C standard libraries

- You can compile C code with C++
  - Weird things can happen, but we aren't going into those subtle issues
- However, you now know and love the standard C libraries
- You can use them in C++ too
- You just have to include different header files

C Library Header	C++ Equivalent	Purpose
`ctype.h`	`cctype`	Character manipulation
`limits.h`	`climits`	Constants for integer limits
`math.h`	`cmath`	Math functions
`stdio.h`	`cstdio`	C I/O functions
`stdlib.h`	`cstdlib`	Random values, conversion, allocation
`string.h`	`cstring`	Null-terminated string manipulation
`time.h`	`ctime`	Time functions

Structs in C++

- A **`struct`** in C++ is treated like a class where all the members are public
- You can even put methods in a **`struct`** in C++
- Otherwise, it looks pretty similar
- You don't have to use the **`struct`** keyword when declaring **`struct`** variables
  - Except in cases when it is needed for disambiguation

Example

- Here's a **`TreeNode`** struct in C++

```
struct TreeNode
{
	int value;
	TreeNode* left;
	TreeNode* right;
};
```

- Write a tree insertion with the following signature

```
void insert(TreeNode* &root, int data);
```

Upcoming

Next time…

- OOP in C++
- C++ madness
- Templates in C++

Reminders

- Start working on Project 6