# Programming with C++

22C:109:SCB
Greg Nichols
Mondays / Wednesdays
6:30 - 7:45(ish)

# Logistics

- Introductions
- Go over syllabus
- About programming and languages
- A little history
- About C++ as a language
- How to write and compile a simple program
- How to write programs in general

# What is Programming?

- Computers are dumb - making them do useful things involves telling them *exactly* what to do, step by step.
- Programming is the process of taking a program and breaking it down into a sequence of steps the computer can handle (CPU instructions).
- CPU instructions are *very* simple: e.g. add two numbers, fetch a number from memory, see if a number is equal to zero, etc.

# Low Level Language

- Usually this refers to assembly code
- Each code instruction becomes a single CPU instruction
- Hard to write, hard to read, hard to maintain
- Not portable between different CPU families
- ... but great for control freaks, and can result in very efficient code

# High Level Languages

- Most programming languages are high-level (C++, C, Perl, PHP, Java, Pascal, Whirl, Python...)
- Easy to write, easy to read (easi*er*, anyway)
- Not (usually) machine specific
- Must be translated into assembly language
  - ... meaning it often loses something in efficiency

# A Bit ‘o History

- Before C++ came C, in the early 1970s
- C was originally invented (no kidding) to play Space Travel, a video game
- Developed along with UNIX
- C was designed to be minimalist:
  - Easy to compile into fast machine code
  - Nothing “behind the scenes”
  - Low level access to memory

# ...and then came C++

- C++ is an extension to C, from the early 1980s
- C++ is "C with classes", and a few extra language features
- ... but still with most of the low-level-ness of C (no hand-holding)
- Fast, powerful, standardized, and very popular
- ... but error prone if you are not careful!

# C++ is:

- **Compiled** (translated into machine code in advance, before run-time)
- **Strongly typed**, meaning that each variable has a type associated with it (float, int, whatever)
- **High level**... but still pretty low
- **Portable** - the same code can often be compiled on many different kinds of computers with little or no modification

# Writing a C++ program

- Programs can be written in any text editor
- On the lab machines try gedit, nedit, emacs, KDevelop, etc.
- Via SSH (linux.cs.uiowa.edu) try pico
- Use whatever text editor or platform or compiler you are most comfortable with, but your program **must** compile on Linux using g++!

# Compiling/Running it

- 1. Compile with g++
- 2. If it works, run the resulting executable
- 3. Like this:

```
[gbnichol@serv16 ~/cpp]$ ls
main.cpp
[gbnichol@serv16 ~/cpp]$ g++ -o program main.cpp
[gbnichol@serv16 ~/cpp]$ ls
main.cpp program
[gbnichol@serv16 ~/cpp]$ ./program
Hello world: 42
[gbnichol@serv16 ~/cpp]
```

```
// a sample program, by Greg
#include <iostream>
using namespace std;

int main()
{
   int baz = 2;
   int foo = 21;
   int result;

   // multiply some stuff
   result = foo * baz;

   // output the result
   cout << “Hello world: ”
        << result << endl;

   return 0;
}
```

# Errors!

- Errors are problems with your program
- Different kinds of errors:
  - Compiler errors
  - Linker errors
  - Runtime errors

# Compiler Errors

- Compiler errors are problems with your code that result in it not compiling
- Code errors, typos, spelling errors, etc.
- Errors must be fixed before the code will compile; warnings don't *have* to be fixed (but you should probably fix them anyway, if you can)

# Linker Errors

- Each cpp file is compiled into an *object file*, which contains the compiled version of that code
- All the object files are "linked" together into a single executable program
- If the object files don't mesh together well (missing functions, duplicate functions, etc.) you get linker errors
- These must be fixed before you can run your program

# Runtime Errors

- You know... bugs!
- Anytime your program crashes or in general doesn't work correctly
- Divide by zero, running out of memory, or just doing the wrong thing

# Errors

- Finding and fixing errors can be tricky and sometimes frustrating - some errors can be hard to find (= vs == for example)
- Solution: practice and be patient.
- The only way to get good at this is to do lots of it!

# Thoughts on Programming

- Programming is the process of taking a program and breaking it down into a sequence of steps you can put into code.
- This is not always easy.
- Doing it *well* requires patience and practice.
- It's fun, though. Really. :-)

# Programming (in general)

- Divide the project up into small chunks.
- Write each chunk independently. Use comments to document anything that needs it.
- *Test* that chunk. Make sure it works.
- *Then* move onto other chunks.
- Compile early and often, and fix any errors and warnings before moving on.