Module 02 - Programming Abstractions

Intro to Computer Science 1 - C++
Professor Scott Frees

Binary Programming

Theoretically, you could program entirely in 1's and 0's. You'd go insane, and accomplish little...

ENIAC (Electronic Numerical Integrator and Computer): (1945)

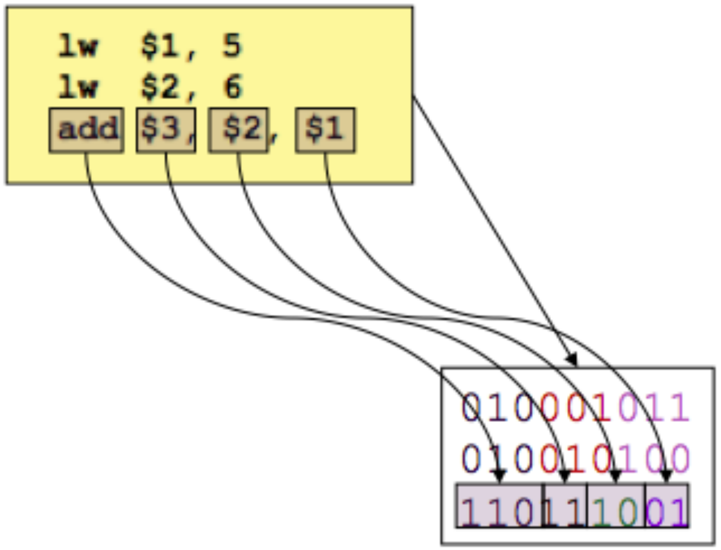
- 19,000 Vacuum Tubes
- 1,500 sq. ft.
- 30 tons

Programming consisted of re-wiring the machine!

Translators

Computers speak binary - we speak in "symbols"

- Assembly language
  - Symbolic Notation for binary instruction



```
lw   $1, 5
lw   $2, 6
add  $3, $2, $1
```

```
010001011
010010100
110111001
```

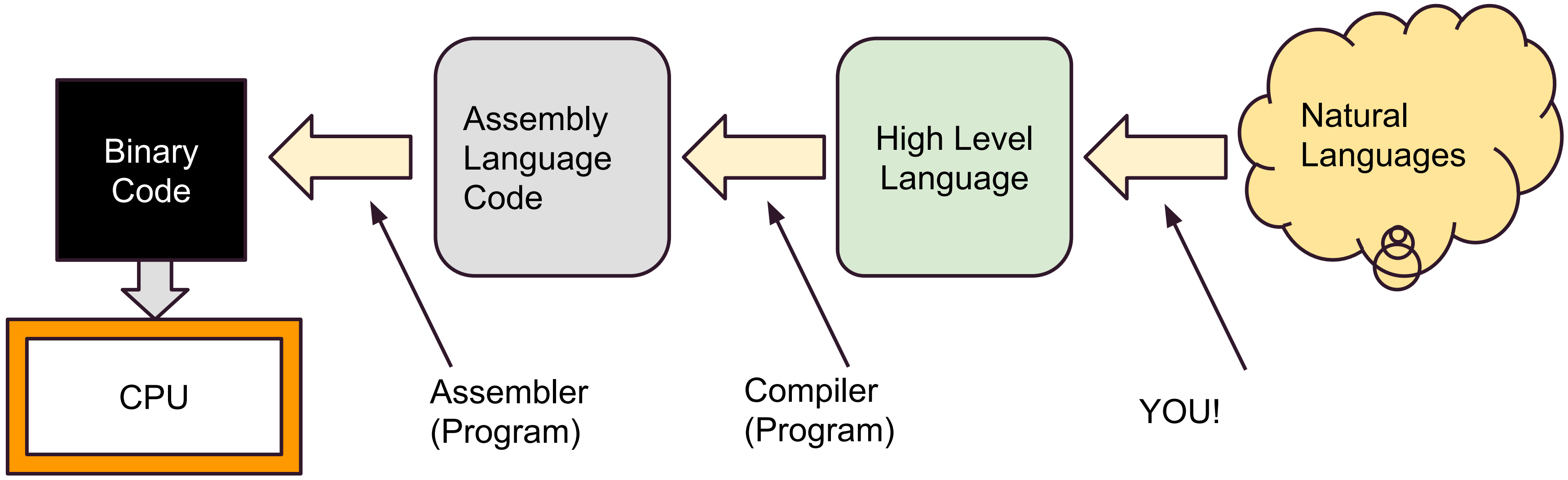
Higher Level Languages

- Assembly language is really just a 1:1 translation
  - Instead of speaking in 1's and 0's, we tell the computer to add/sub/load/store.
- The problem is that when we want to solve complex problems, working at the "machine" level is incredibly tedious
- For example:
  - "Upload this image and tag me so my friends see this come up on their news feed"
  - Really does convert to many add/sub/load/store - but we don't want to think like this...

High Level Languages

High level languages add concepts like:

- Variable names
- Conditionals *(if this is true, do that, otherwise do this…)*
- Loops *(keep doing this until that happens)*
- And many more elaborate structures.



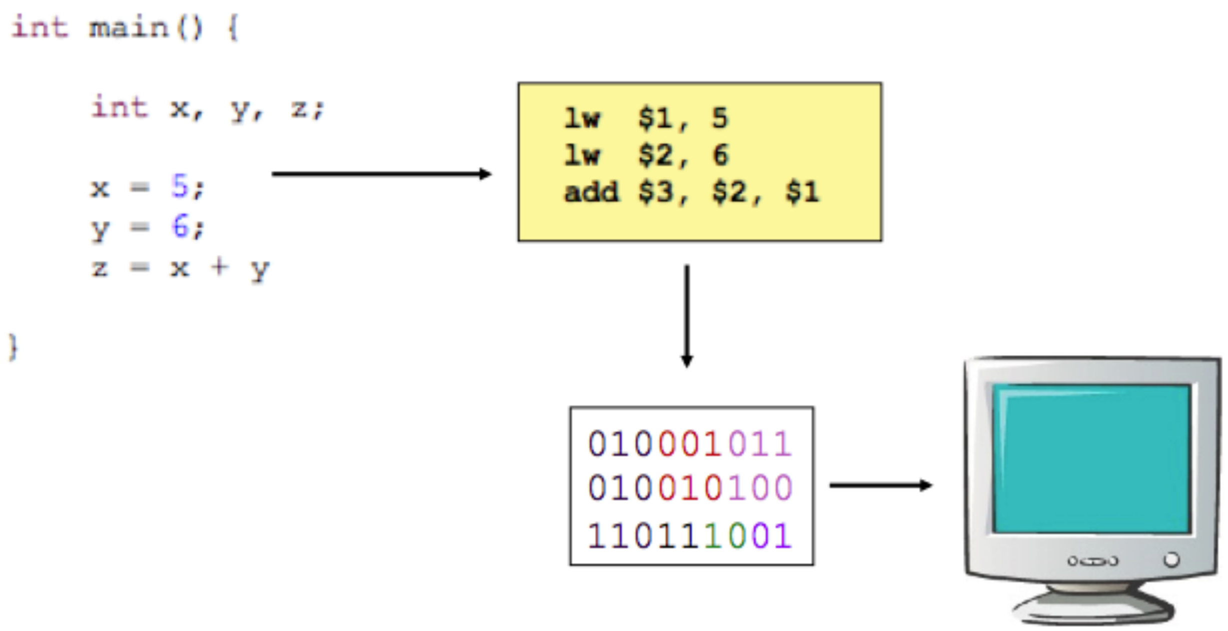
Higher Level Languages

C, C++, Java, C#, JavaScript, Ruby, Python, Perl, PHP, COBOL, Groovy, Fortran, HTML, CSS, XML, XSL, Visual Basic, R, Haskell, Clojure...

There is probably another being created right now...

Some are very specialized, some are more general purpose. Once you learn one, its generally much easier to learn the others!

Example: C++



```
int main() {

    int x, y, z;

    x = 5;
    y = 6;
    z = x + y

}
```

```
lw  $1, 5
lw  $2, 6
add $3, $2, $1
```

```
010001011
010010100
110111001
```

Turning C++ into Binary

In this class we will use Microsoft Visual Studio as our “compiler”

We will create C++ source code (stored in files with .cpp extension)

Visual studio will “build” an executable file with binary instructions (with a .exe extension)

Visual Studio / Command Line

We will now go slowly through Visual Studio and the command line

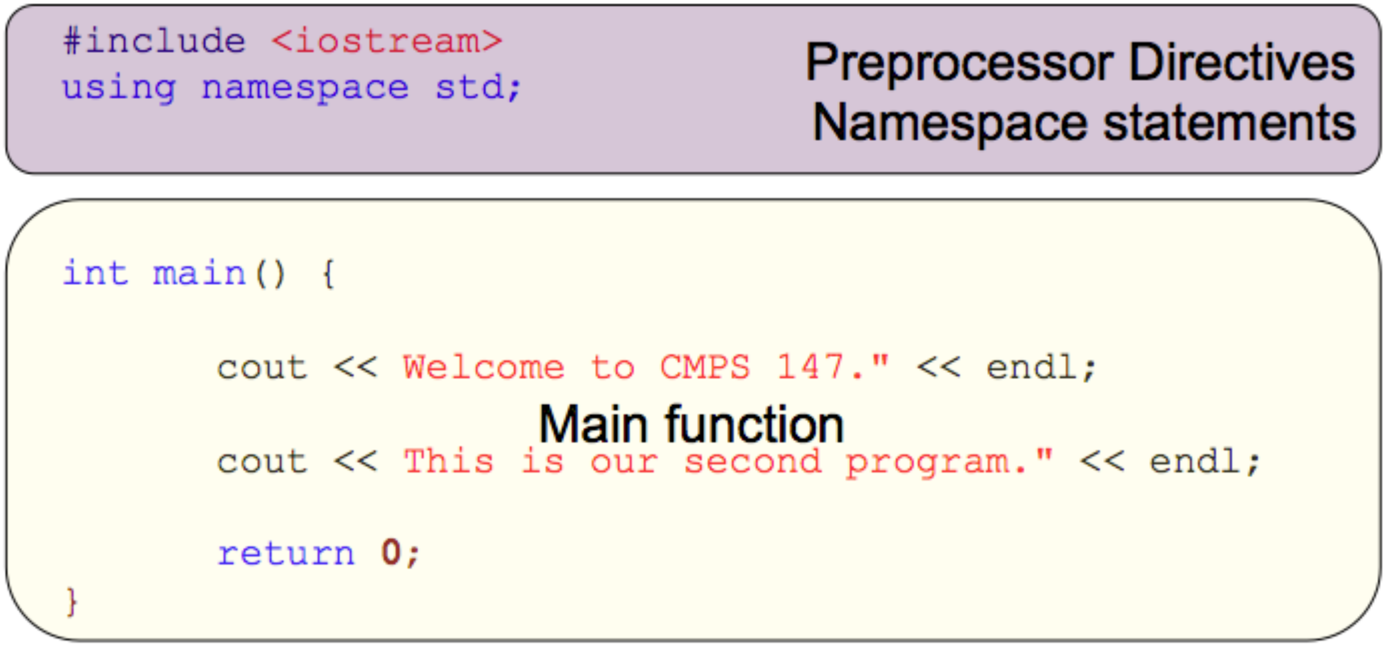
See Moodle for more details walkthroughs

This will be difficult at first - but I promise you will get the hang of it in a few days!

A first C++ Program

```
#include <iostream>
using namespace std;
int main() {
  cout << “Welcome to CMPS 147." << endl;
  cout << “This is our first program." << endl;
  return 0;
}
```

Parts of a C++ program



```
#include <iostream>
using namespace std;
```

```
int main() {

        cout << Welcome to CMPS 147." << endl;

        cout << This is our second program." << endl;

        return 0;
}
```

Pre-Processor Directives

`#include` directives tell the compiler where to find additional source code referenced in your program

`cout` is an output command, defined in `iostream` - a C++ library (lots of C++ code).

Ignore namespace statement <u>for now, </u>we will discuss this later.

Inside the “main” function

The *main* function begins with its declaration

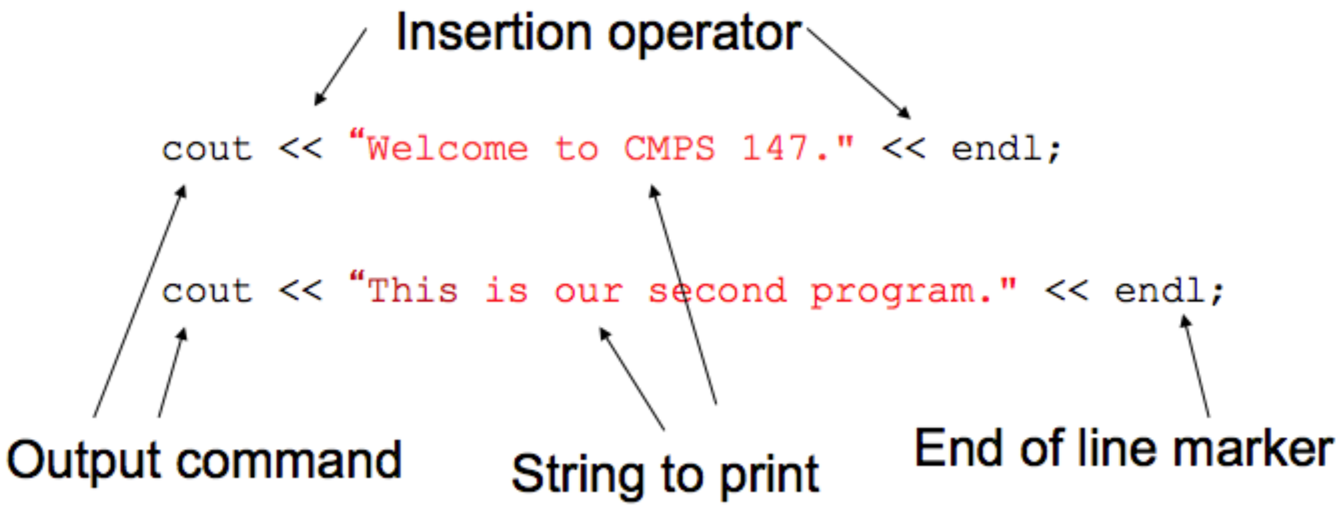
```
int main()
```

Its bounds are defined by curly brackets **{ }**

Within the main function are *executable statements (instructions)*.

- Statement Syntax
  - All statements end with a ‘;’
  - Good practice to keep one per line.

Printing - primer



```
cout << “Welcome to CMPS 147." << endl;

cout << “This is our second program." << endl;
```

Think of << as separating items to be printed - fields

Comments

You must always document your code.

- Assist a *human* reader in understanding what your program does.
- At a minimum, should always have your name, date, and “purpose”.
- Single line comments `//   comment`
- Multi-Line comments **/* comment */**

Lab 1

Create a **new Visual Studio Project**

Write a C++ program to print out your name

For next class

Download Visual Studio at home

Get this program to run!

Read Chapter 2 in the text book