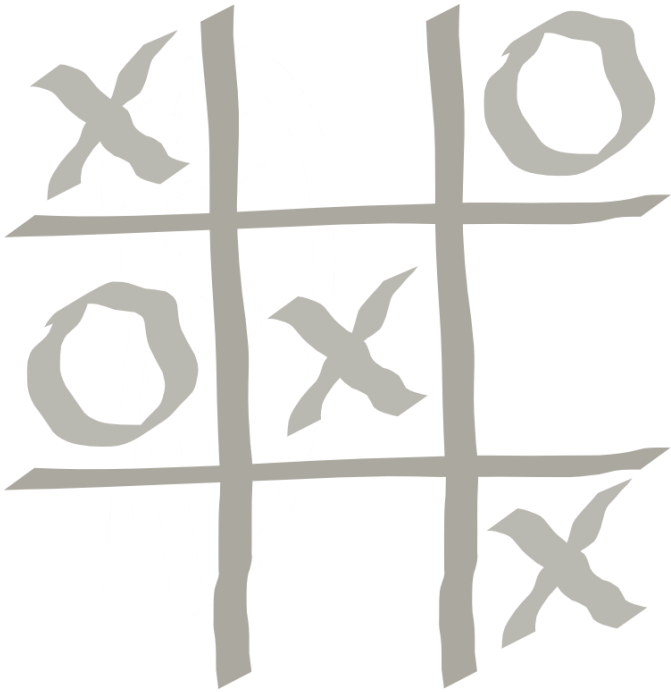
Introduction; Simplest Program; Basic Output

ITP 165 – Fall 2015
Week 1, Lecture 1

Computers are much smarter than us, right?

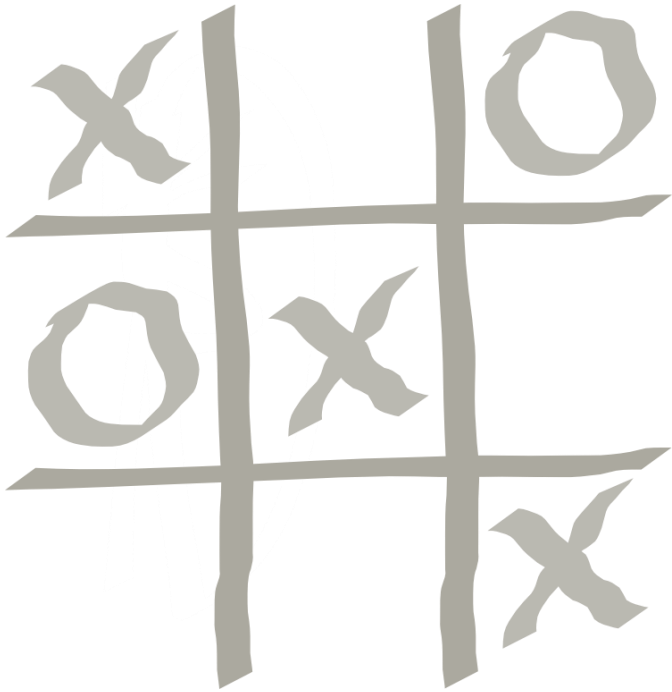
Explaining Tic-Tac-Toe to a Human

- It's a 2 player game
- There is a grid of 9 squares
- First player is Xs and Second player is Os
- When it's a player's turn, they draw 1 X or O
- First player to get 3 symbols in a line wins.
- (If you play perfectly you always will draw).



Explaining Tic-Tac-Toe to a Computer

- It's a 2 player game
- There is a grid of 9 squares
- First player is Xs and Second player is Os
- When it's a player's turn, they draw 1 X or O
- First player to get 3 symbols in a line wins.
- (If you play perfectly you always will draw).

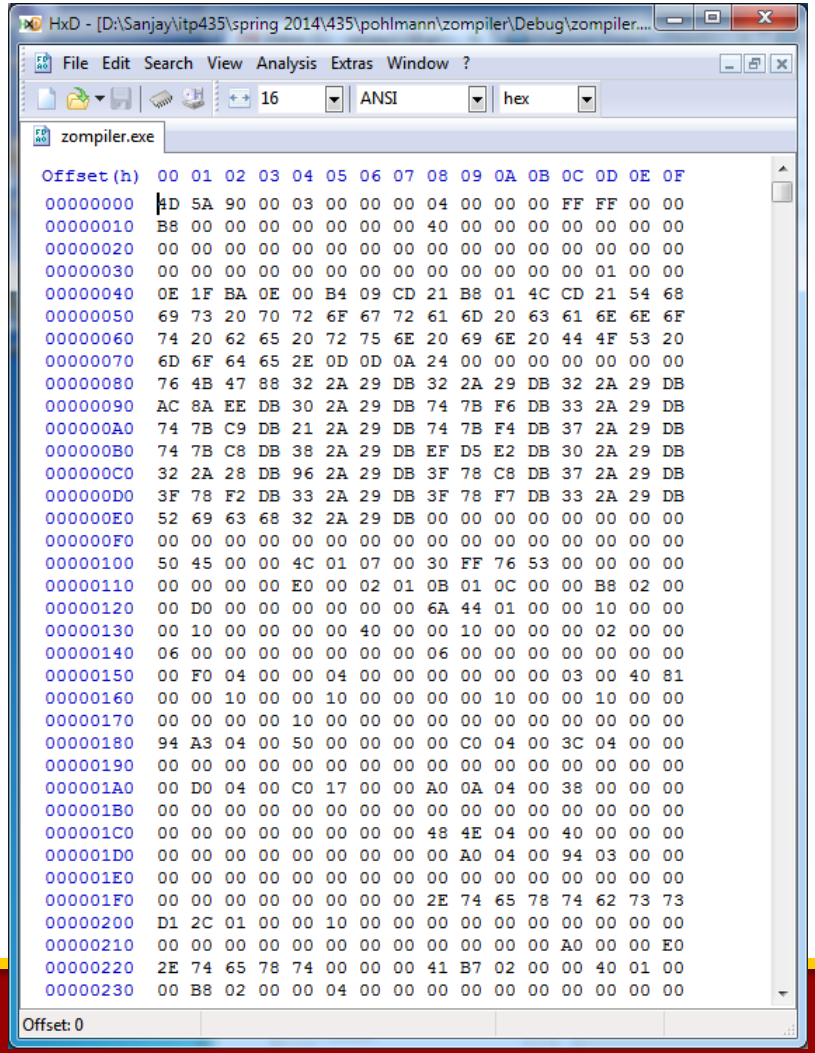


Computers Understand…

- Basic numbers
- Basic arithmetic
- Basic logic
- That's it!
- *BUT…*
- They can do billions of such operations every second

Machine Code

- Computers actually only understand machine code



High-Level Programming Language

- A programming language that abstracts the low-level machine code details
- May have some words that look like English
- Examples:

Who uses C++?


Microsoft


facebook


Bloomberg


NORTHROP GRUMMAN

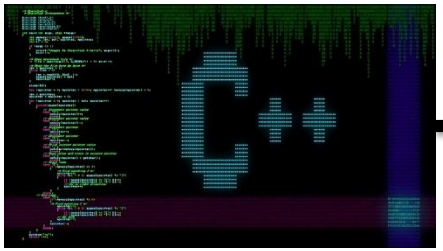

Adobe

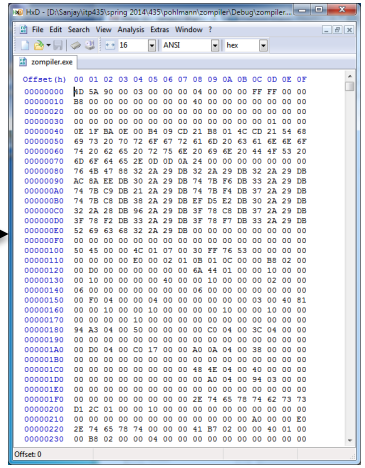

BLIZZARD
ENTERTAINMENT


Google

Compiler

- Behind the scenes converts our C++ code into machine code





Before we start programming…

- The most important thing is not the syntax (grammar) of C++…

- In order to be successful at programing, you must understand the idea of an **algorithm** – a “step-by-step procedure for calculations”*

- Before you start writing one line of code, you should plan out step-by-step what your program must do

A simple problem…

- Given a deck of cards…

- …group them by suit

Card Grouping Algorithm #1

1. Throw all of the cards on the ground.
2. Pick up all of the cards, and hope that they are grouped by suit.
3. If they aren't grouped by suit, repeat steps 1-2 until they are grouped by suit.

Card Grouping Algorithm #2

1. Create four piles, one for each suit.
2. Go through each card one by one, and place it in the corresponding pile.
3. Once you have ran out of cards in the deck, you will have four fully grouped piles.

So what's this prove?

- It's important to have a good algorithm before you actually start writing the code

- You could write flawless code for algorithm #1, ***but because algorithm #1 is a terrible algorithm, the program will be terrible***

- Before you write one line of code, figure out the logical steps you will need to follow in order to solve the problem

Now let's look at some C++

- Don't be afraid to ask questions!

The Simplest C++ Program

```
int main()
{
    return 0;
}
```

The “body” of the program

```
int main()
{
    return 0;
}
```

Statement

- A statement in C++ is a single command

- Basic statements must end with a semicolon.

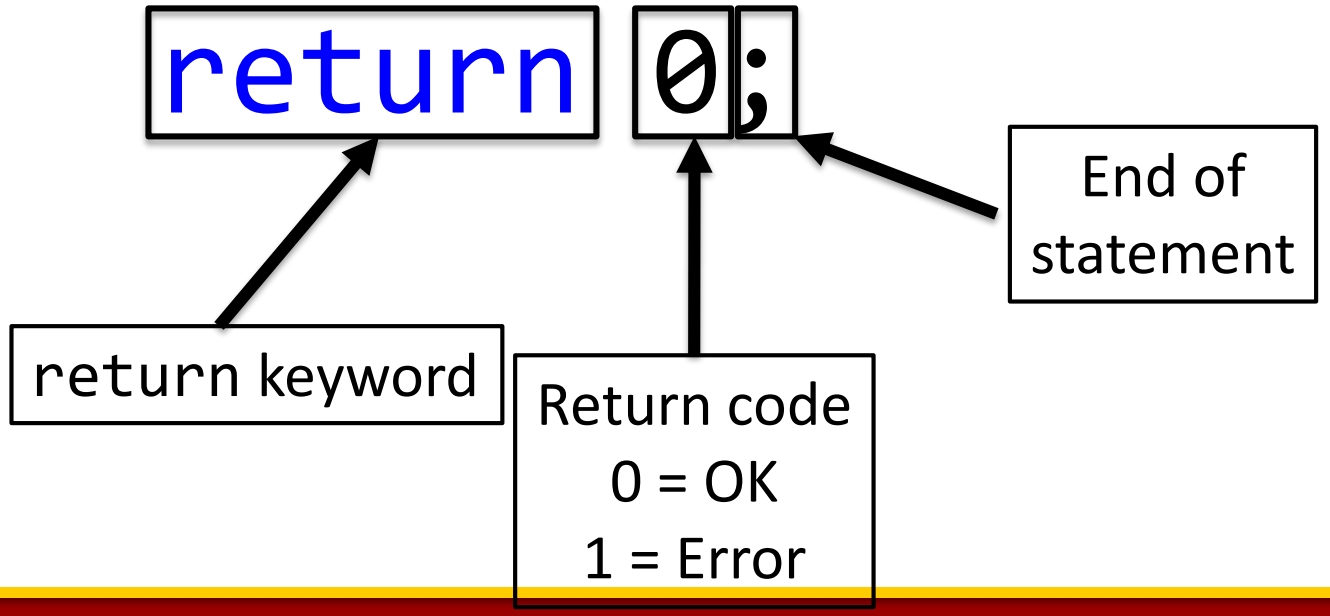
- So in our first example, we have the following statement:

```
return 0;
```

“Return” statement

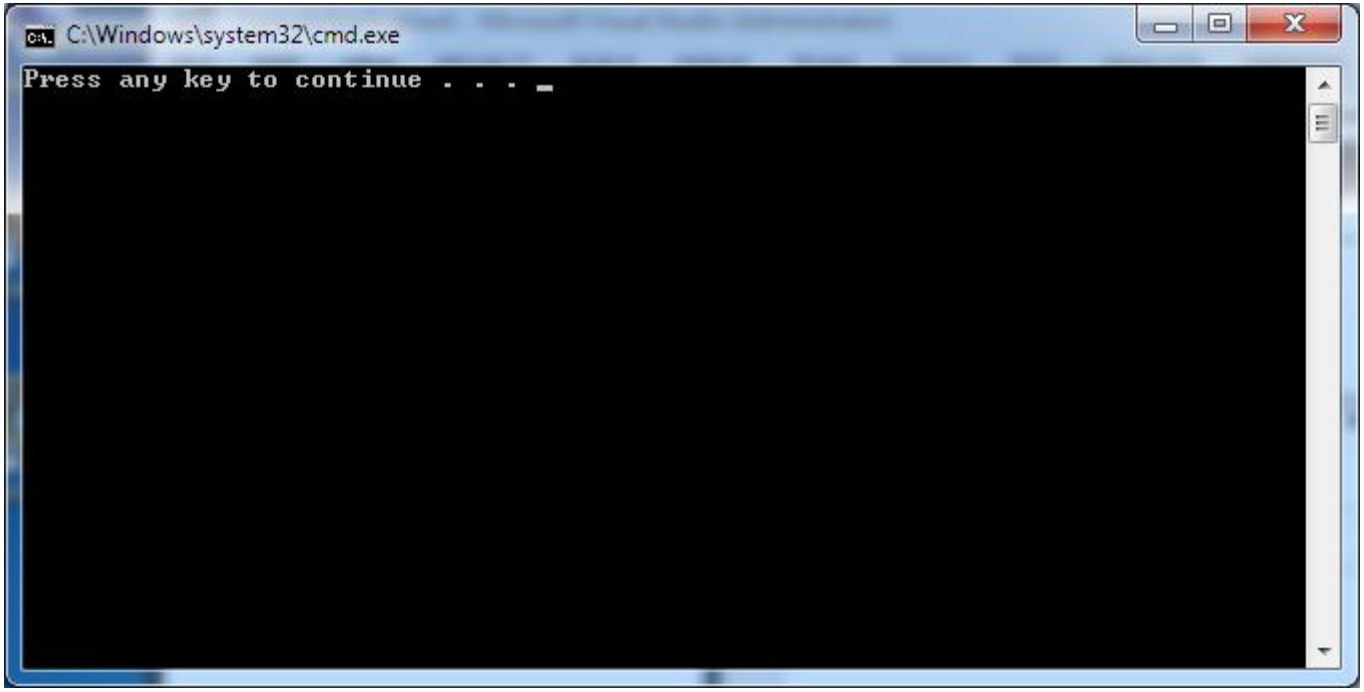
- The return statement says “okay the program is finished”*
- Your program must always end in a return statement

- Syntax (eg. the grammar) for return statement:



What happens if we run this program?

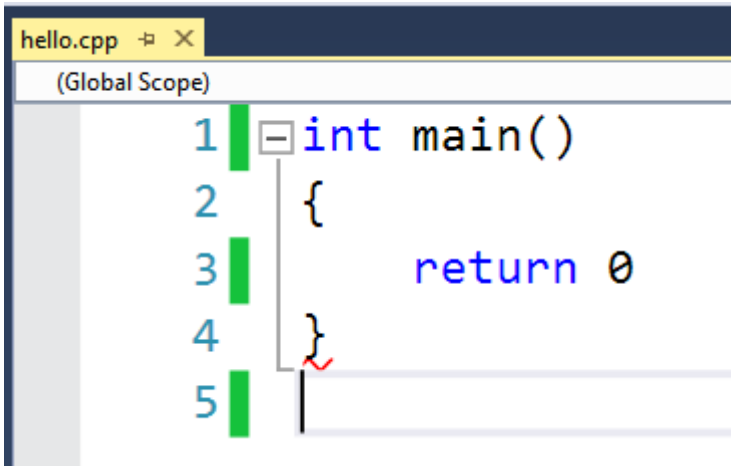
```
int main()
{
    return 0;
}
```

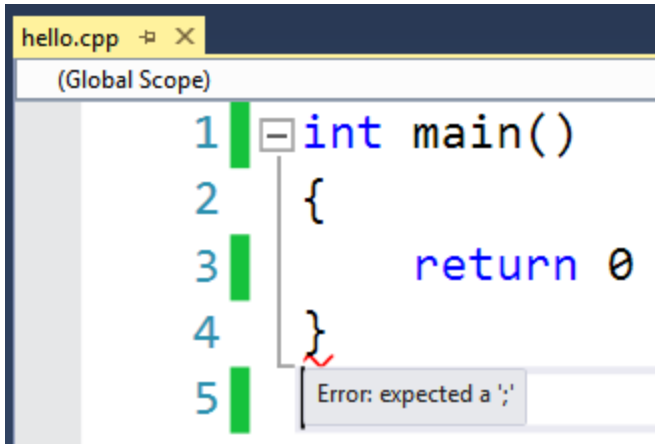


Syntax Errors

- The syntax of C++ is very rigid; make a mistake and you will get an error

- Visual Studio and Xcode will try to help you find errors





Build Error

- If you make a mistake in the program, when you try to run in Visual Studio it will say:

- Always say **NO** and find/fix the error(s)!

Comment

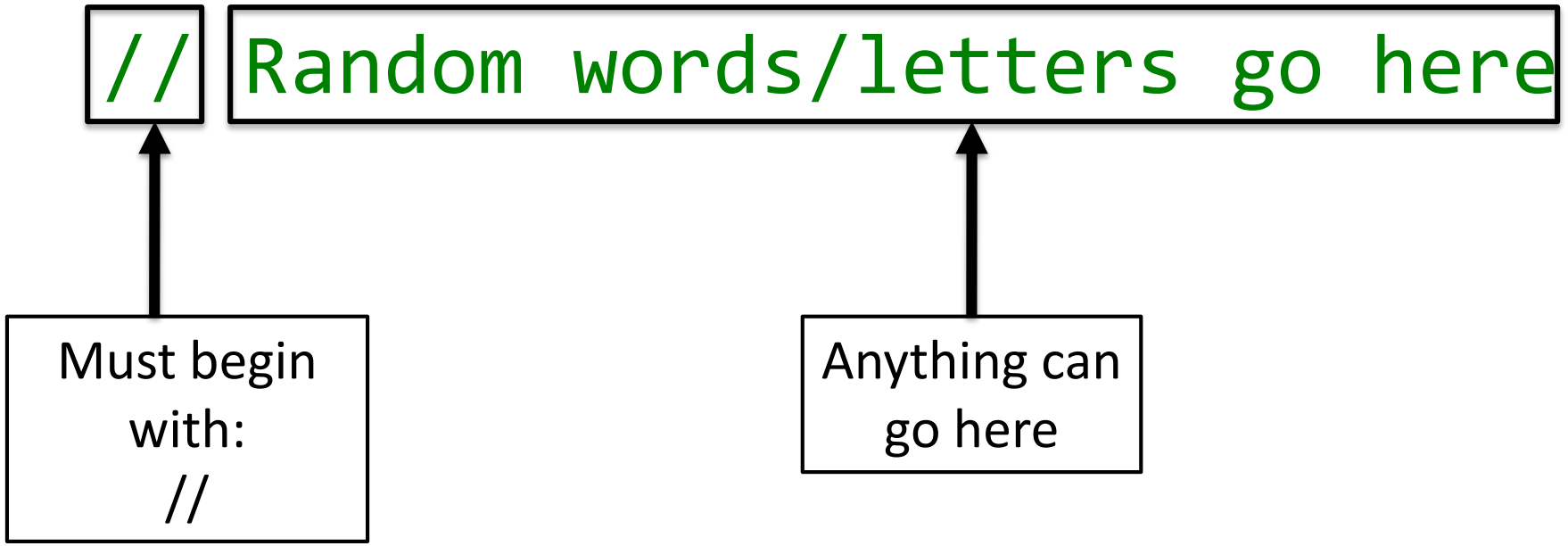
- A comment is a note you can leave in the program
- It is not code that “runs”

- Comments start with // and end at the end of the line

```
// This is a comment
```

Comment Syntax

- It's pretty simple:

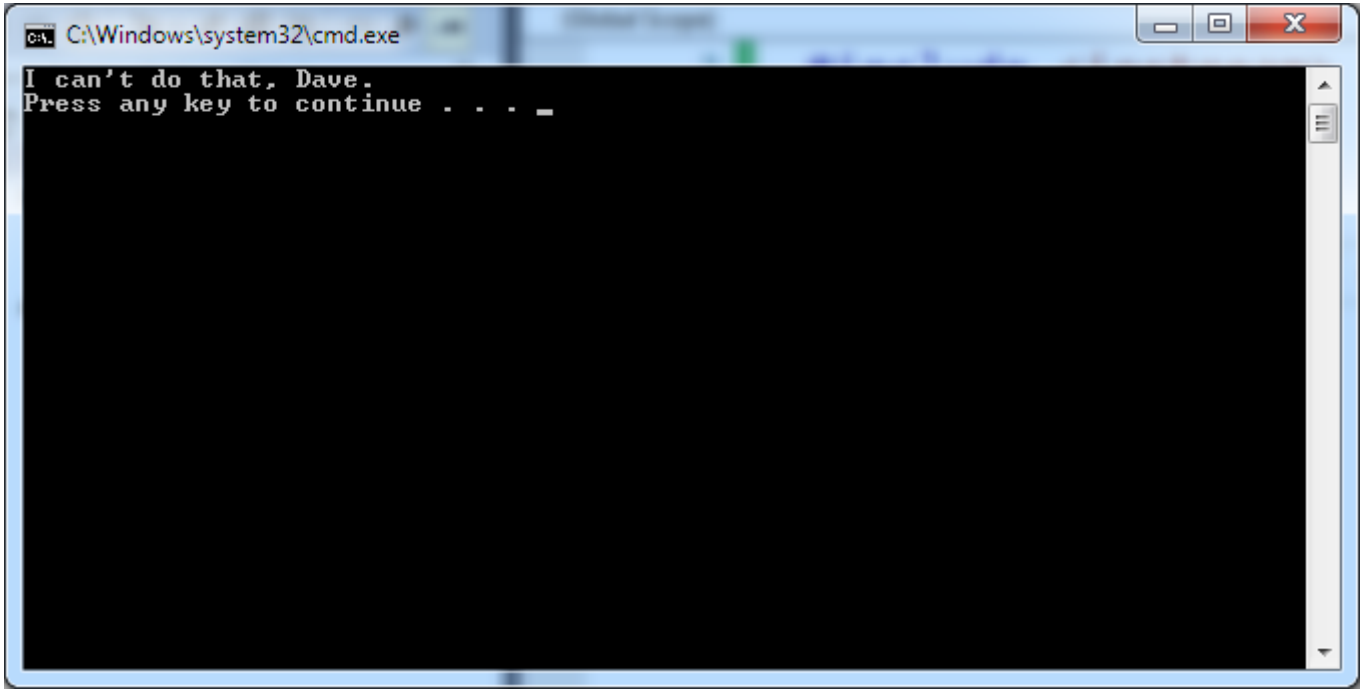


Simplest Program, Now with comments!

```
int main()
{
    // This is my awesome program
    return 0;
    // Program done!!
}
```

Basic Text Output

- Let's make a program that does something *slightly* more exciting:



Code for Basic Text Output

```
// Basic output example
#include <iostream>
int main()
{
    std::cout << "I can't do that, Dave." << std::endl;
    return 0;
}
```

Code for Basic Text Output – What's new?

```
// Basic output example
#include <iostream>
int main()
{
    std::cout << "I can't do that, Dave." << std::endl;
    return 0;
}
```

#include Directive

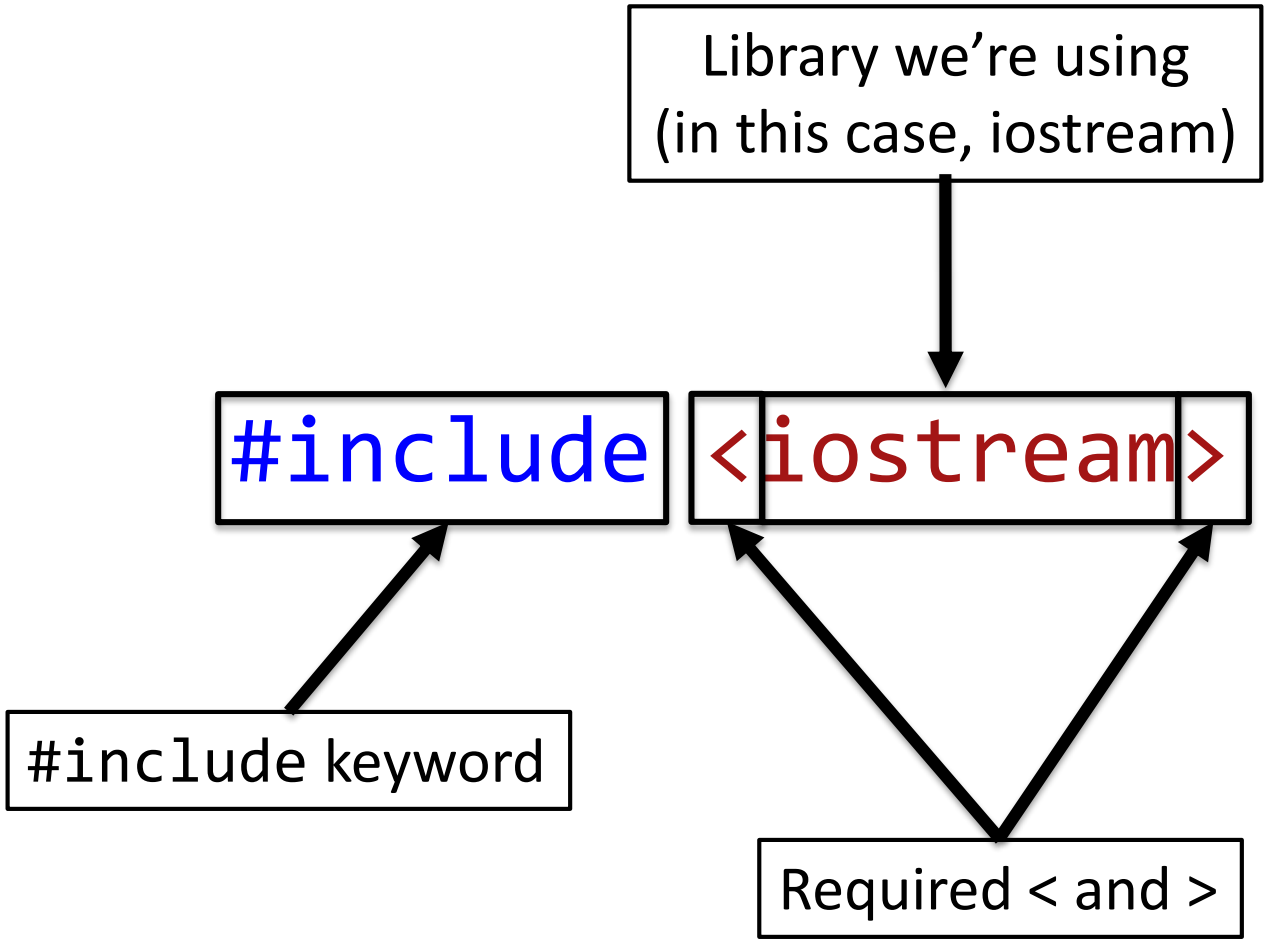
- C++ has a lot of different things in the language
- `#include` is required to specify what parts of the programming language you're using (called a ***library***)
- Note that many basic things (such as `return`, arithmetic, etc.) do not require a `#include`
- Slightly more complex stuff (like text input and output) does require specific `#include` directives

#include Directive, cont'd

- `#include` directives ***always*** go in the "header" part of the program, ***before*** the "int main" line

```
// Basic output example
#include <iostream>
int main()
{
   std::cout << "I can't do that, Dave." << std::endl;
   return 0;
}
```

#include Directive Syntax



C++ Libraries

- There are a lot of different libraries we might want to `#include`

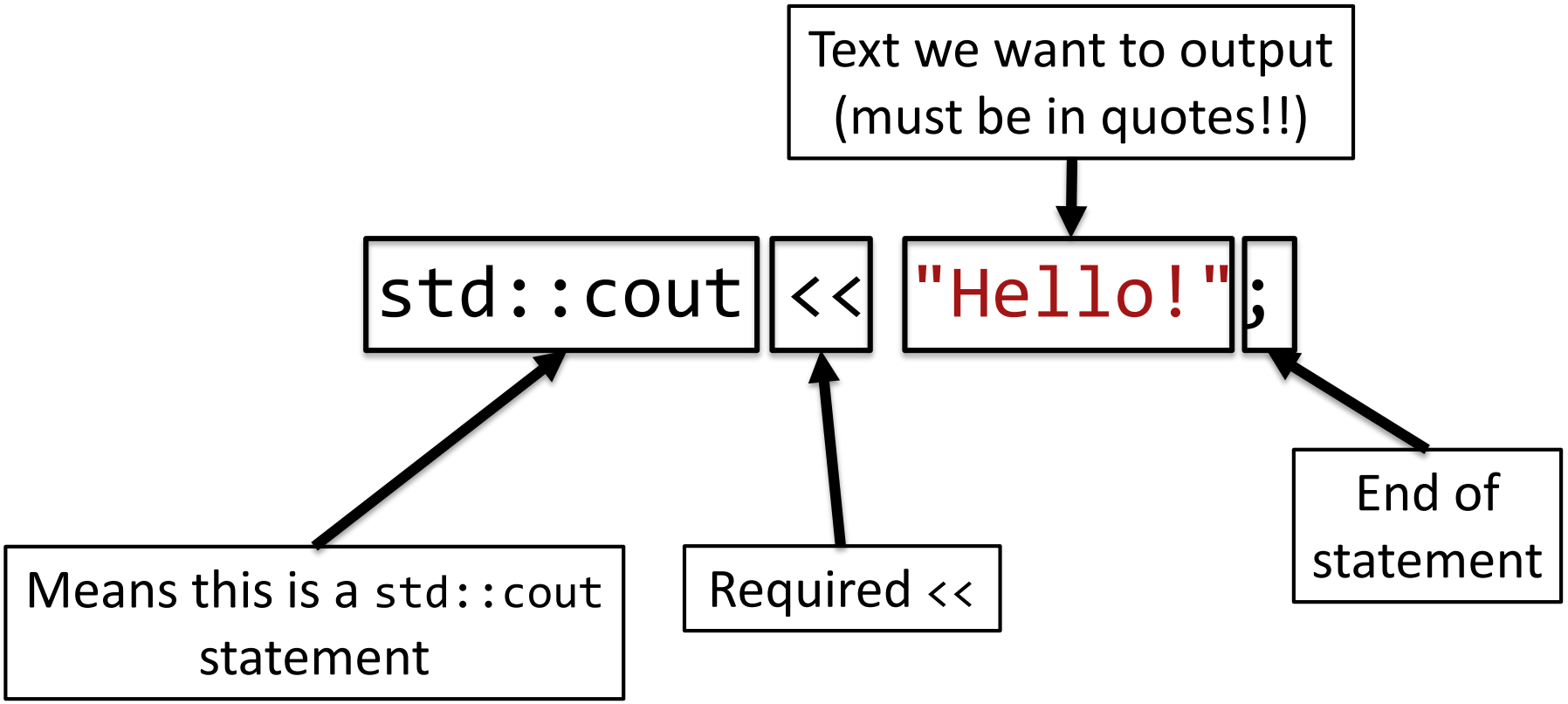
- For now, we will only be using two:
  - `iostream` – Allows for basic text input and output
  - `string` – More on this in a little bit

What about that weird “std::cout” statement?

```
// Basic output example
#include <iostream>
int main()
{
   std::cout << "I can't do that, Dave." << std::endl;
   return 0;
}
```

std::cout statement – For "console" output

- Simplest version of std::cout would look like this:



std::cout Chaining

- We can chain multiple phrases by adding on additional << prior to the semicolon

- For example, these two std::cout statements would both output the same thing:

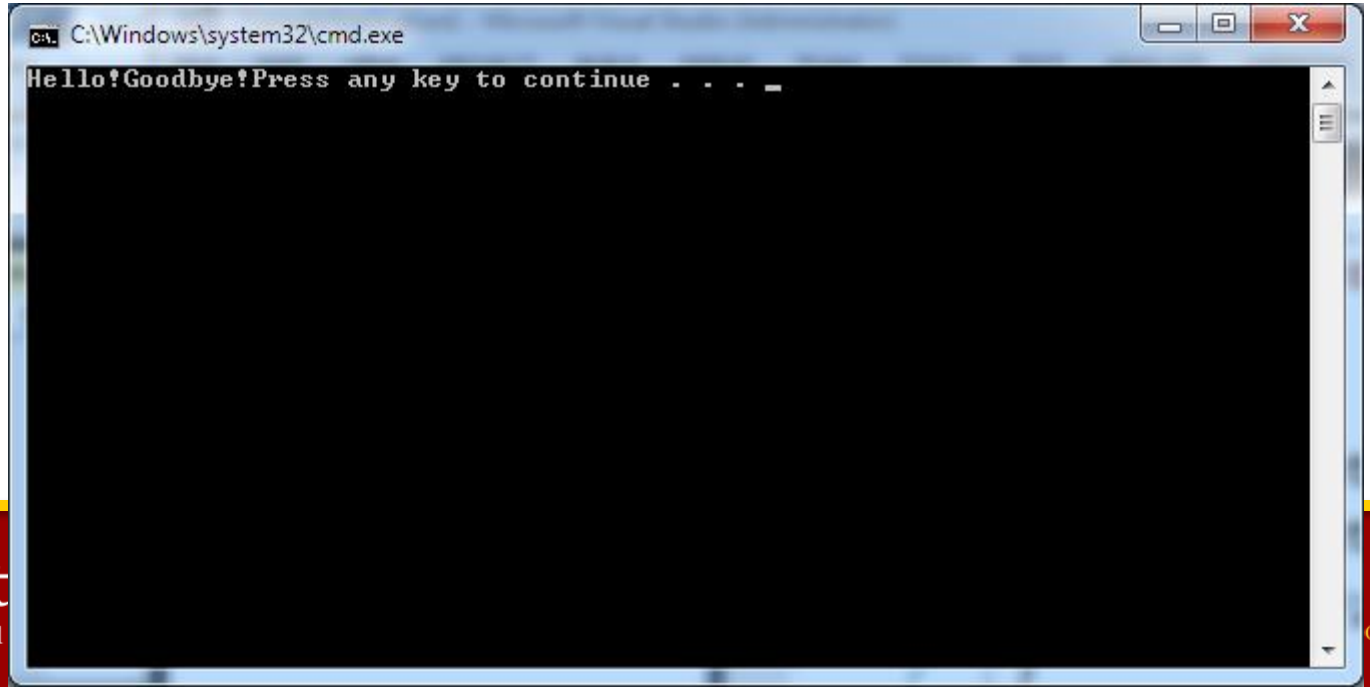
```
// Outputs "Hello world!"
std::cout << "Hello world!";

// Also outputs "Hello world!"
std::cout << "Hello " << "world!";
```

Chain this!

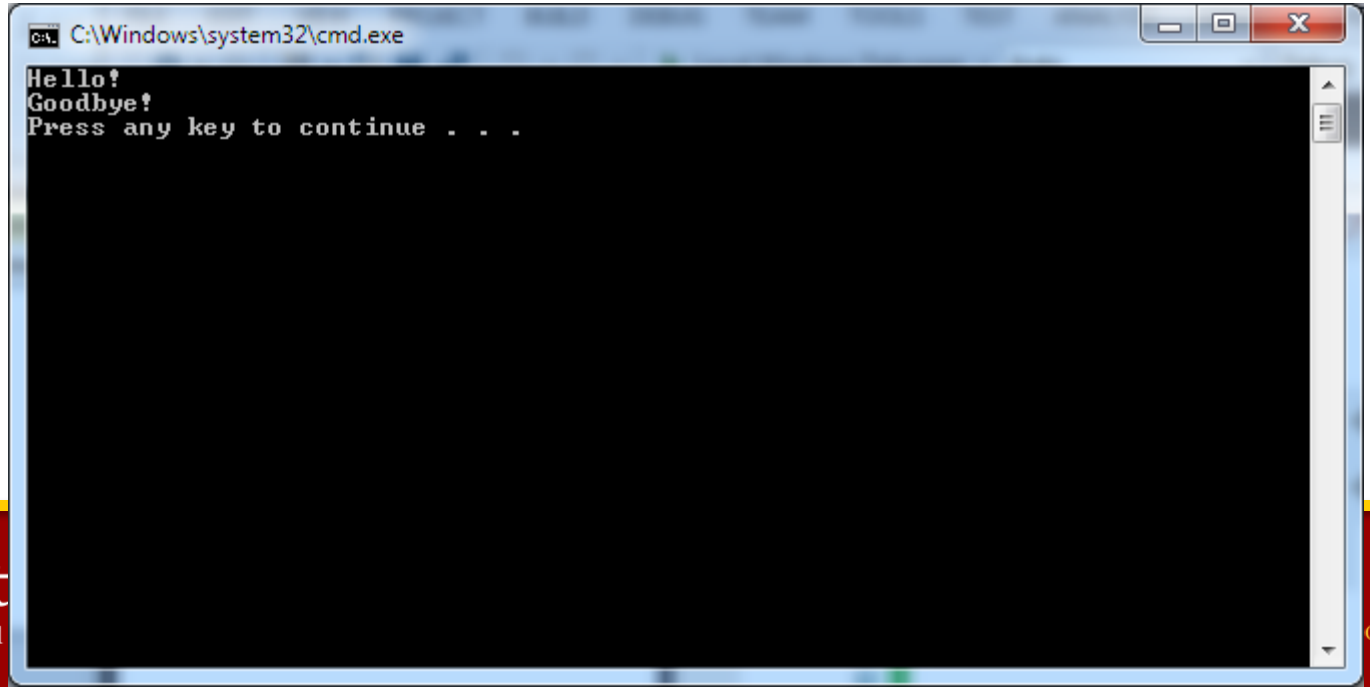
What about two std::cout statements?

```
#include <iostream>
int main()
{
    std::cout << "Hello!";
    std::cout << "Goodbye!";
    return 0;
}
```



std::endl

```
#include <iostream>
int main()
{
    std::cout << "Hello!" << std::endl;
    std::cout << "Goodbye!" << std::endl;
    return 0;
}
```



Going back to the earlier example…

```
// Basic output example
#include <iostream>
int main()
{
    std::cout << "I can't do that, Dave." << std::endl;
    return 0;
}
```