



# Introduction; Simplest Program; Basic Output

ITP 165 – Fall 2015  
Week 1, Lecture 1



# Computers are much smarter than us, right?

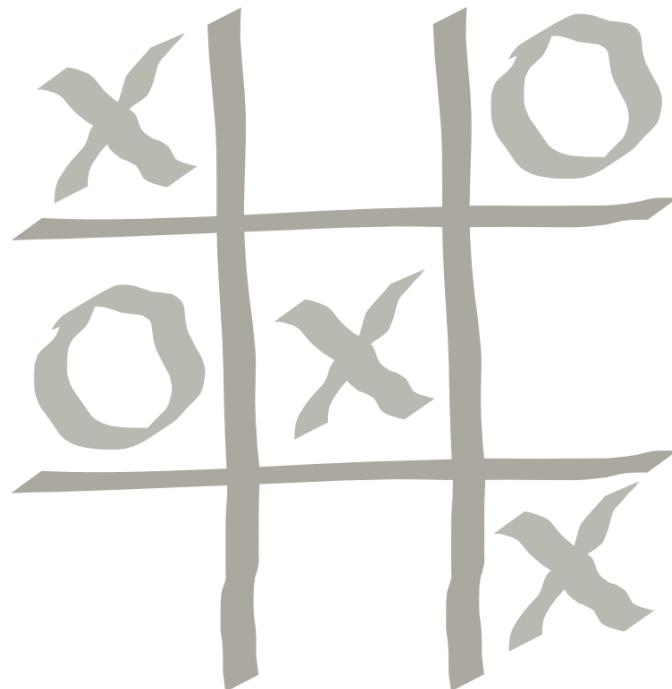
**WRONG!**





# Explaining Tic-Tac-Toe to a Human

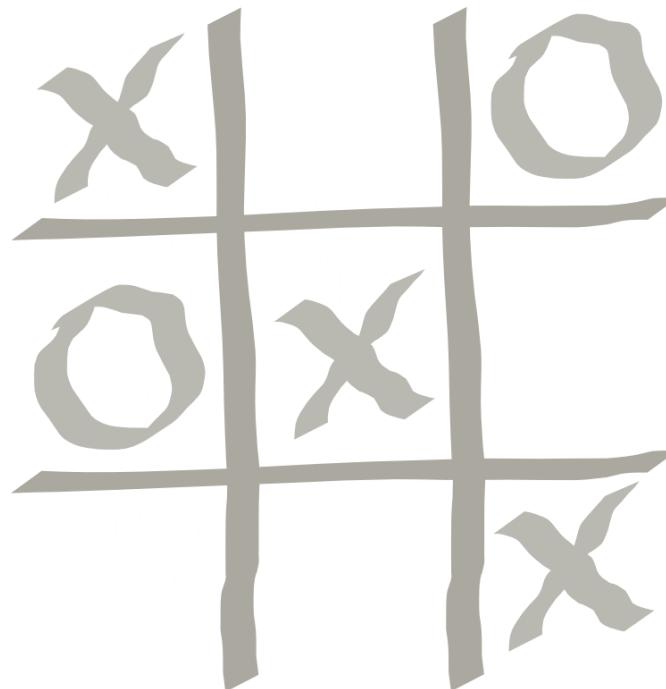
- It's a 2 player game
- There is a grid of 9 squares
- First player is Xs and Second player is Os
- When it's a player's turn, they draw 1 X or O
- First player to get 3 symbols in a line wins.
- (If you play perfectly you always will draw).





# 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).





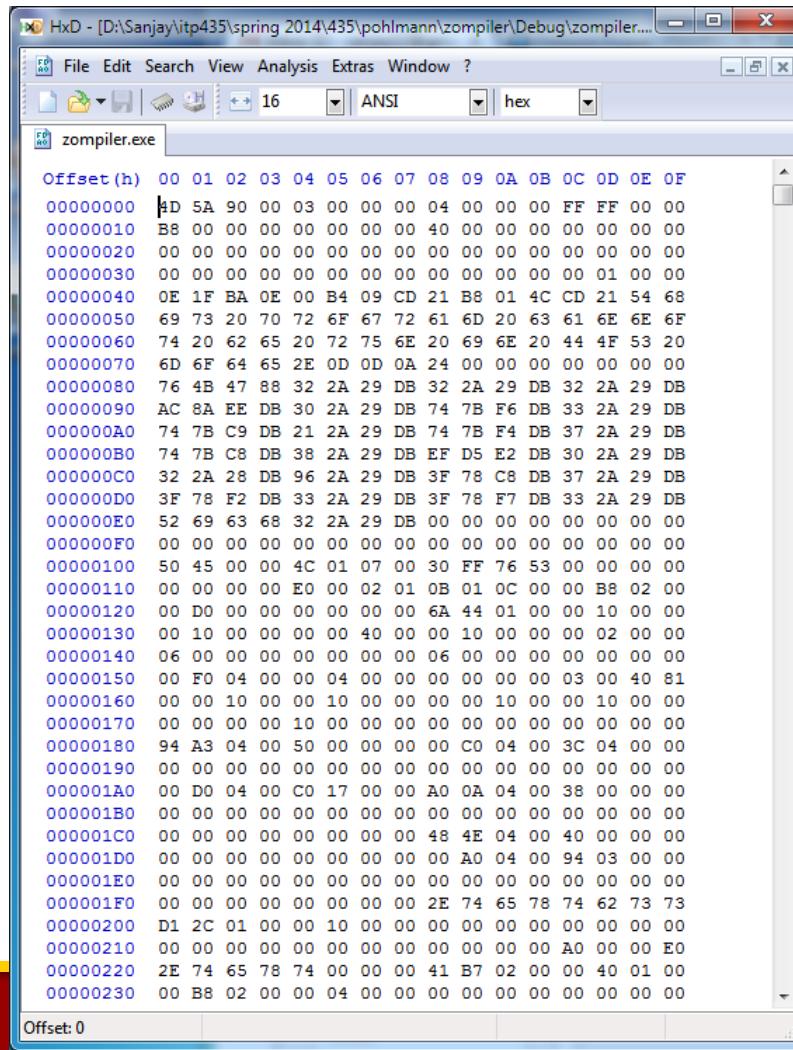
# Computers Understand...

- 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



The screenshot shows the HxD hex editor interface with the file "zompiler.exe" open. The window title is "HxD - [D:\Sanjay\itp435\spring 2014\435\pohlmann\zompiler\Debug\zompiler....]".

The main pane displays memory dump data in a grid format. The columns are labeled "Offset(h)" and "00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F". The data starts with the byte sequence: 4D 5A 90 00 03 00 00 00 04 00 00 00 00 FF FF 00 00.

The data continues with several lines of assembly-like mnemonics and their corresponding hex values:

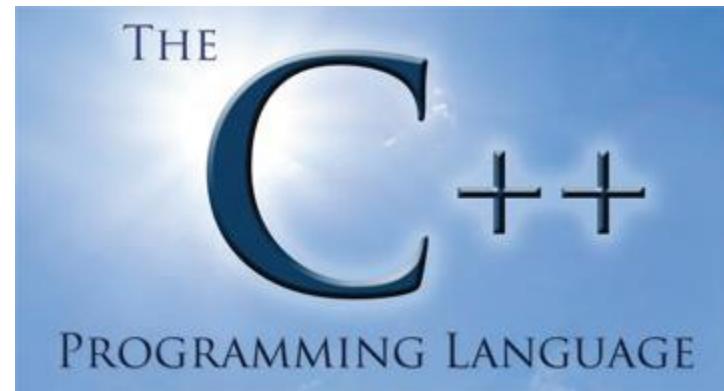
- B8 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00
- 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
- 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
- 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
- 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00
- 76 4B 47 88 32 2A 29 DB 32 2A 29 DB 32 2A 29 DB
- AC 8A EE DB 30 2A 29 DB 74 7B F6 DB 33 2A 29 DB
- 74 7B C9 DB 21 2A 29 DB 74 7B F4 DB 37 2A 29 DB
- 74 7B C8 DB 38 2A 29 DB EF D5 E2 DB 30 2A 29 DB
- 32 2A 28 DB 96 2A 29 DB 3F 78 C8 DB 37 2A 29 DB
- 3F 78 F2 DB 33 2A 29 DB 3F 78 F7 DB 33 2A 29 DB
- 52 69 63 68 32 2A 29 DB 00 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 50 45 00 00 4C 01 07 00 30 FF 76 53 00 00 00 00
- 00 00 00 00 E0 00 02 01 OB 01 OC 00 00 B8 02 00
- 00 00 00 00 00 00 00 00 00 6A 44 01 00 00 10 00 00
- 00 10 00 00 00 00 40 00 00 10 00 00 00 02 00 00
- 06 00 00 00 00 00 00 06 00 00 00 00 00 00 00 00
- 00 F0 04 00 00 04 00 00 00 00 00 00 03 00 40 81
- 00 00 10 00 10 00 00 00 00 10 00 00 10 00 00
- 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
- 94 A3 04 00 50 00 00 00 C0 04 00 3C 04 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 00 D0 04 00 C0 17 00 00 A0 0A 04 00 38 00 00
- 00 00 1B0 00 00 00 00 00 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 48 4E 04 00 40 00 00
- 00 00 1D0 00 00 00 00 00 00 00 A0 04 00 94 03 00
- 00 00 1E0 00 00 00 00 00 00 00 00 00 00 00 00 00
- 00 00 1F0 00 00 00 00 00 00 00 2E 74 65 78 74 62
- 73 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- D1 2C 01 00 10 00 00 00 00 00 00 00 00 00 00 00
- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
- 2E 74 65 78 74 00 00 00 41 B7 02 00 00 40 01 00
- 00 B8 02 00 00 04 00 00 00 00 00 00 00 00 00 00

The bottom status bar shows "Offset: 0".



# 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



Bloomberg

***NORTHROP GRUMMAN***



Google



# Compiler



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



A screenshot of a hex editor window titled 'HxD - [D:\Sangay\tp435\Spring 2014\435\polmann\compiler\Debug\zompiler...]' showing assembly code. The code includes instructions like LD, ST, ADD, SUB, and various memory operations. The assembly code is as follows:

```
Offset(h): 00 03 02 00 04 05 06 07 08 09 03 00 00 00 00 FF FF 00 00  
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 00 00 00 00 FF FF 00 00  
00000010 BB 00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 00 00  
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000040 0E 1F BA 02 00 B4 09 CD 21 88 01 4C CD 21 54 68  
00000050 69 73 20 70 72 EF 67 72 61 6D 20 63 61 6E 6E 6F  
00000060 69 73 20 70 72 EF 67 72 61 6D 20 63 61 6E 6E 6F  
00000070 6D 69 64 65 22 CD 0D 0A 24 00 00 00 00 00 00 00 00  
00000080 76 4B 47 85 32 2A 29 0A 32 2A 29 DB 32 2A 29 DB  
00000090 AC 8A EF DB 30 2A 29 DB 30 2A 29 DB 30 2A 29 DB  
000000A0 74 78 C5 DB 38 2A 29 EF DS E2 DB 30 2A 29 DB  
000000B0 74 78 C5 DB 38 2A 29 EF DS E2 DB 30 2A 29 DB  
000000C0 32 22 28 DB 96 2A 29 DB 3F 78 C8 DB 37 2A 29 DB  
000000D0 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50  
000000E0 52 69 63 63 32 2A 29 DB 00 00 00 00 00 00 00 00 00  
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000100 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50  
00000110 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50 50  
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000130 00 10 00 00 00 00 40 00 00 10 00 00 00 00 02 00 00  
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000150 00 F0 04 00 00 04 00 00 00 00 00 00 00 03 00 40 81  
00000160 00 00 10 00 00 10 00 00 00 10 00 00 10 00 00 10 00 00  
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000180 94 A3 04 00 50 00 00 00 00 00 00 00 04 00 3C 04 00 00  
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 38 00 00  
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 48 1E 04 00 40 00 00  
000001D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 94 03 00  
000001E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
000001F0 00 00 00 00 00 00 00 00 00 00 00 00 2E 74 65 78 74 62 73 73  
00000200 D1 2C 01 00 00 10 00 00 00 00 00 00 00 00 00 00 00 00  
00000210 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00000220 2E 74 65 78 74 00 00 00 41 87 02 00 00 00 40 01 00  
00000230 00 B8 02 00 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00
```



# Before we start programming...

- The most important thing is not the syntax (grammar) of C++...
- In order to be successful at programming, 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

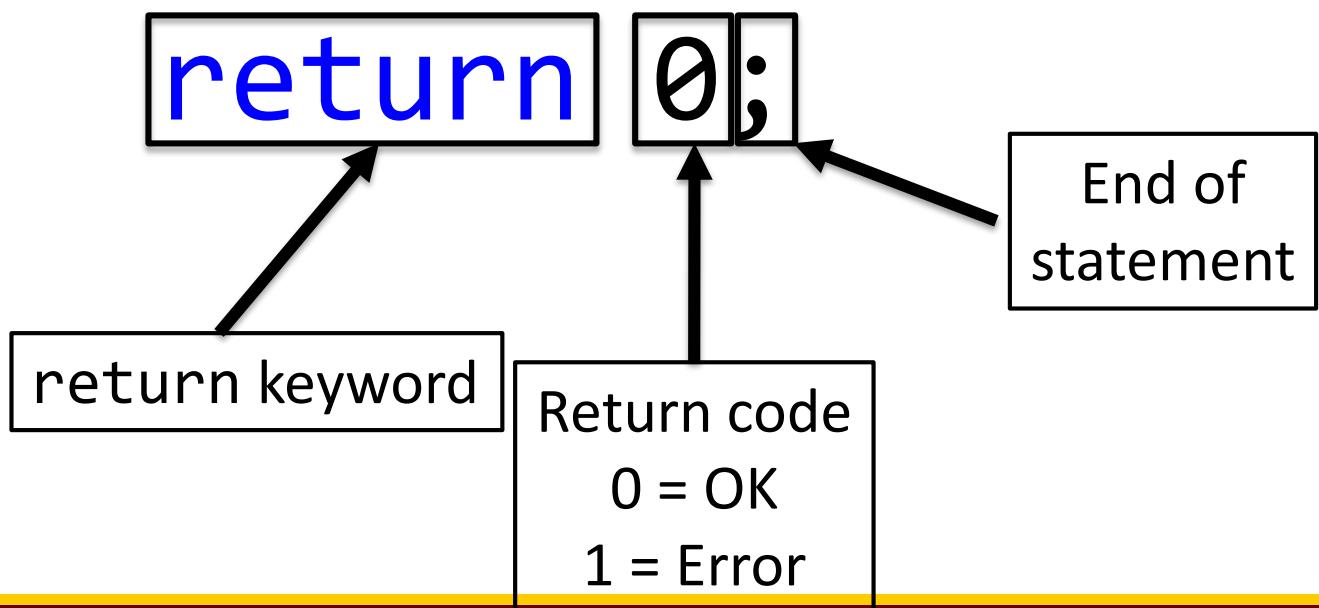
- A statement in C++ is a single command
- Basic statements must end with a semicolon.
- So in our first example, we have the following statement:

```
return 0;
```



# “Return” statement

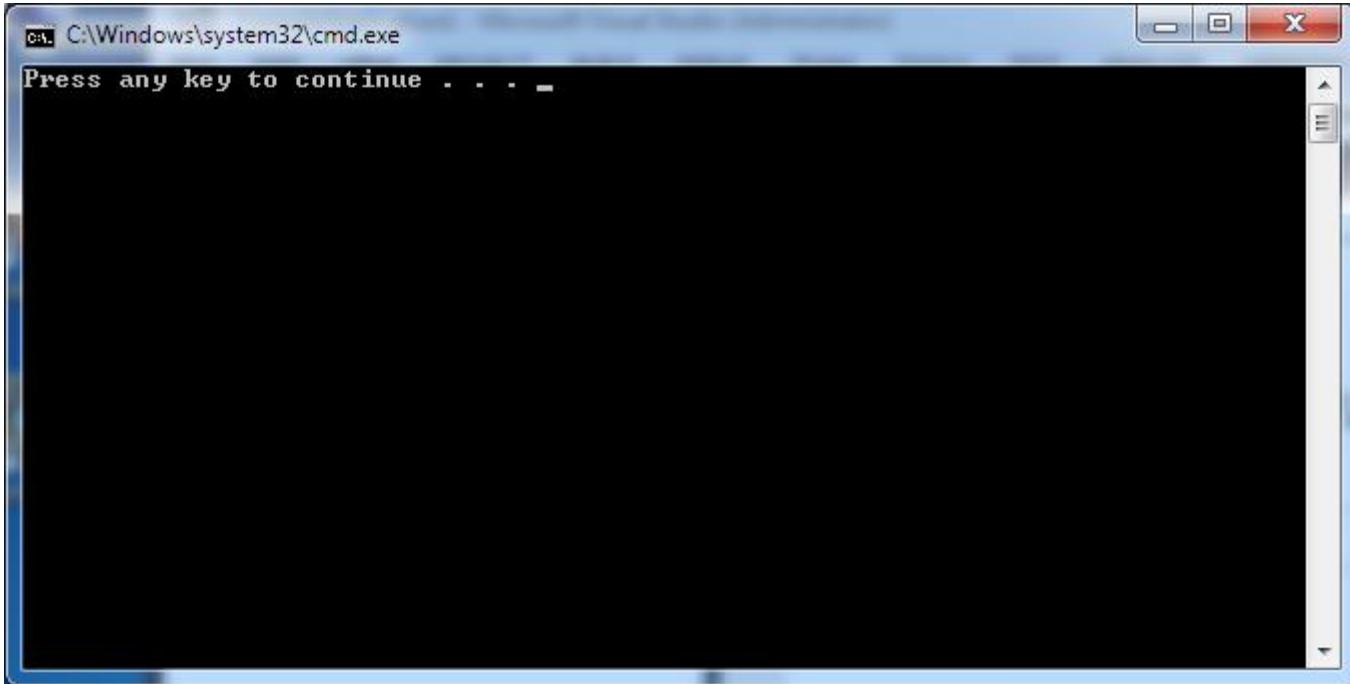
- The return statement says “okay the program is finished”\*
- Your program must always end in a return statement
- Syntax (eg. the grammar) for return statement:





# 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

A screenshot of a Windows-based IDE (Visual Studio) showing a file named "hello.cpp". The code is:

```
1 int main()
2 {
3     return 0
4 }
5 }
```

The cursor is at the end of line 5. A red squiggly underline is under the closing brace of the main function, and a red arrow points from the error message to it. The status bar at the bottom of the IDE window shows "File1.cpp" and "1 error(s)".

A screenshot of the Xcode IDE showing a file named "hello.cpp". The code is identical to the one in the Visual Studio screenshot:

```
1 int main()
2 {
3     return 0
4 }
5 }
```

The cursor is at the end of line 5. A red squiggly underline is under the closing brace of the main function, and a red arrow points from the error message to it. A tooltip box at the bottom right of the screen displays the error message: "Error: expected a ';'".



# 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

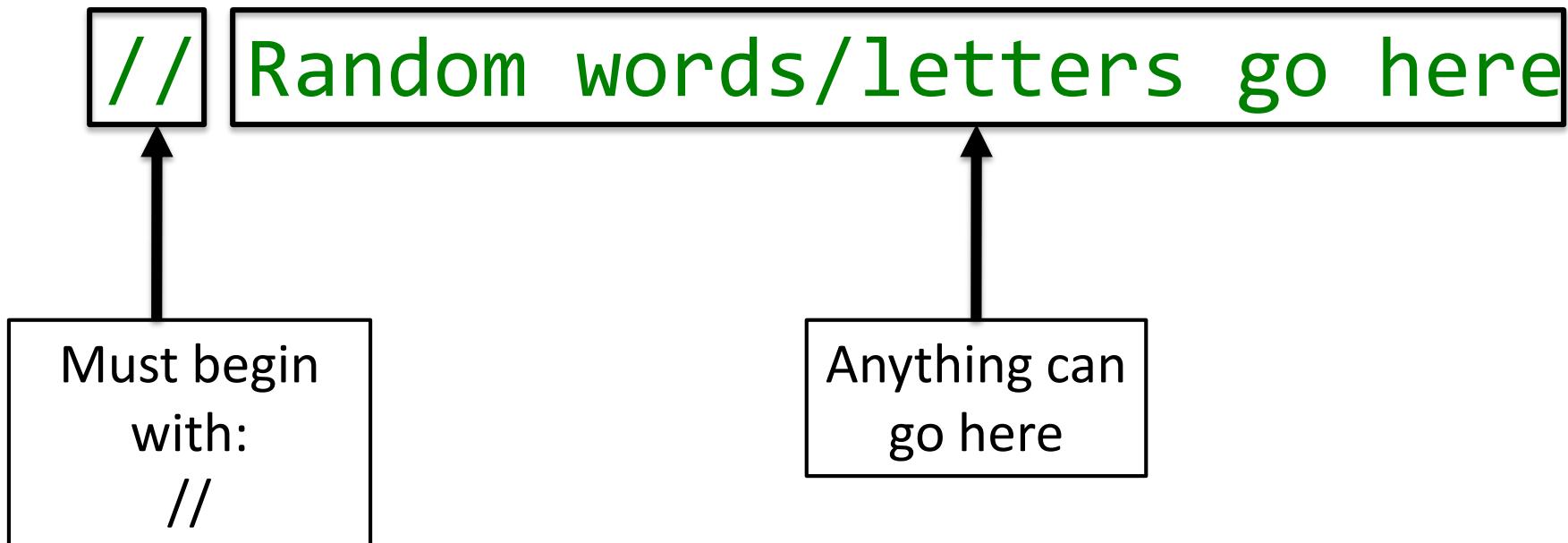
- A comment is a note you can leave in the program
  - It is not code that “runs”
- 
- Comments start with // and end at the end of the line

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



# Comment Syntax

- It's pretty simple:



# Simplest Program, Now with comments!

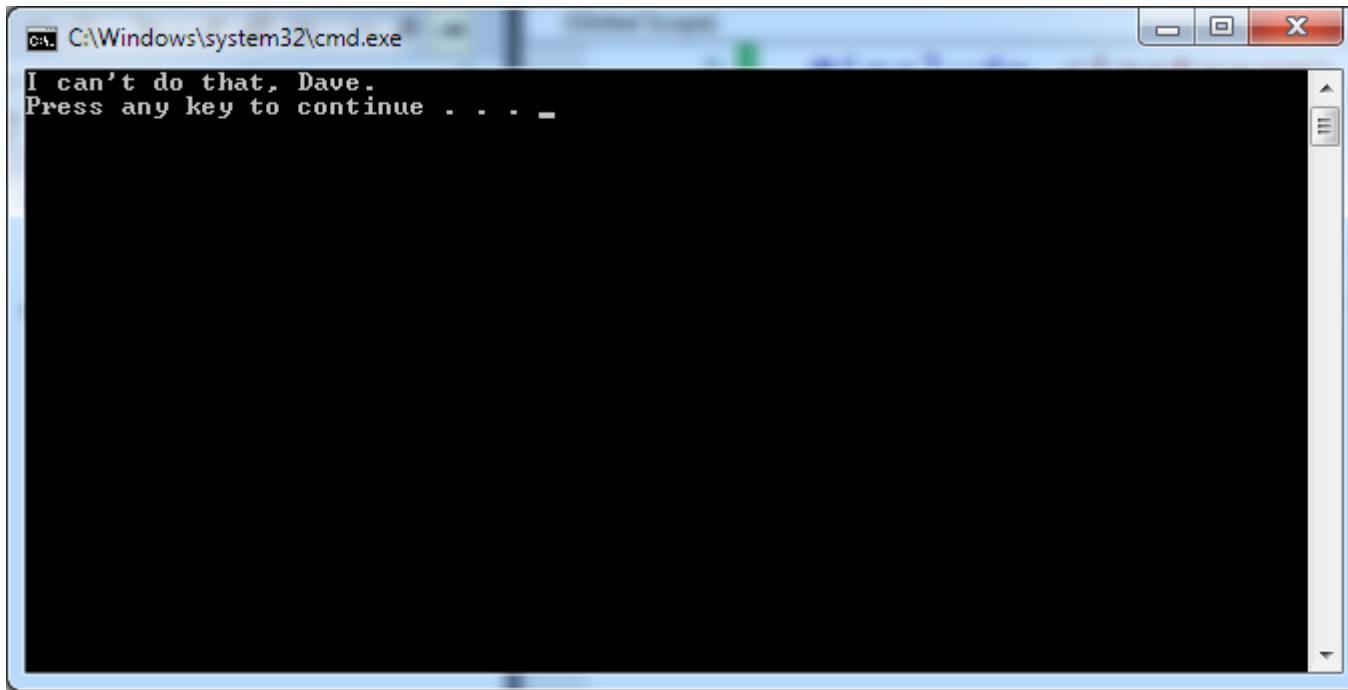


```
int main()
{
    // This is my awesome program
    return 0;
    // Program done!!
}
```



# 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



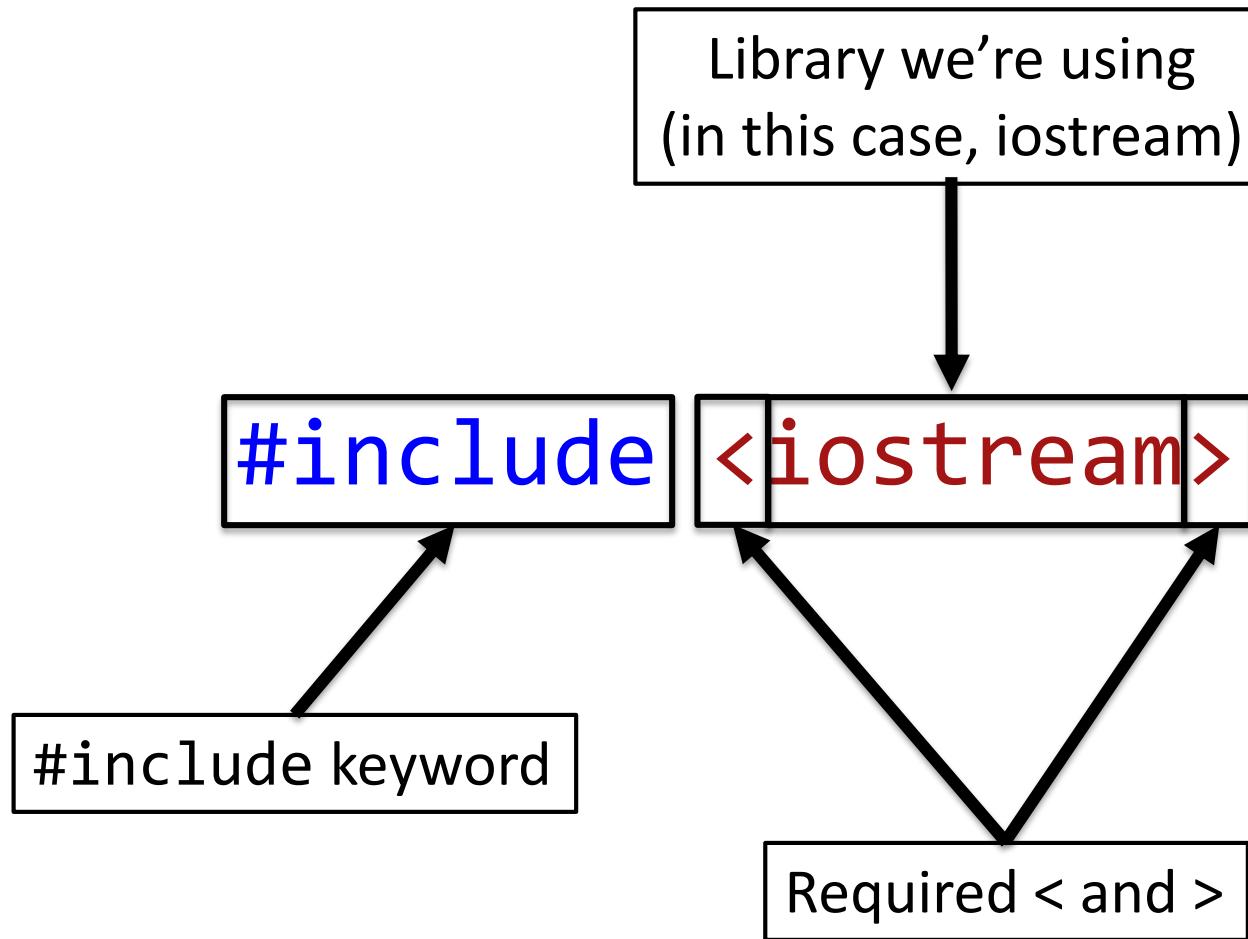
# #include Directive, cont'd

- `#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?

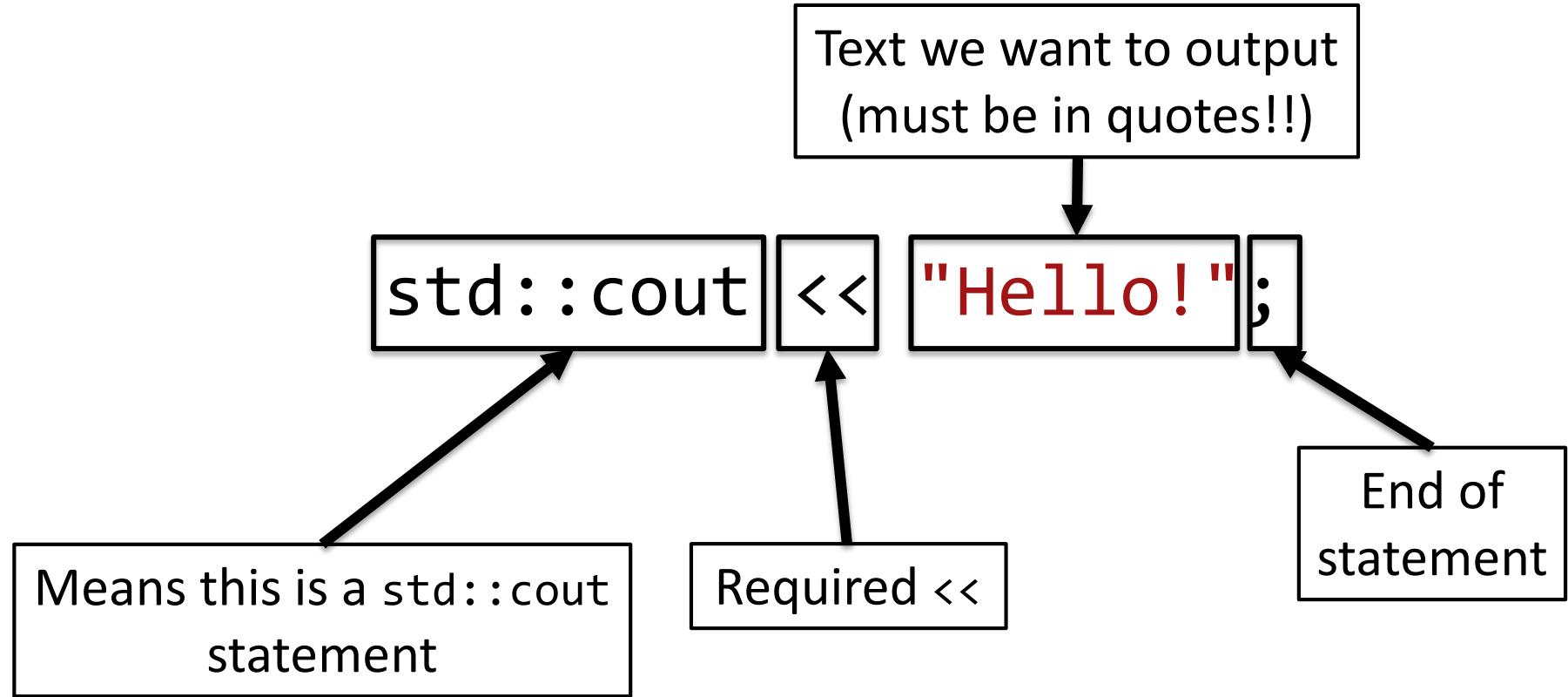


```
// Basic output example
#include <iostream>
int main()
{
    std::cout << "I can't do that, Dave." << std::endl;
    return 0;
}
```



# 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!";
```

<< "world!"

Chain this!



# What about two std::cout statements?

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

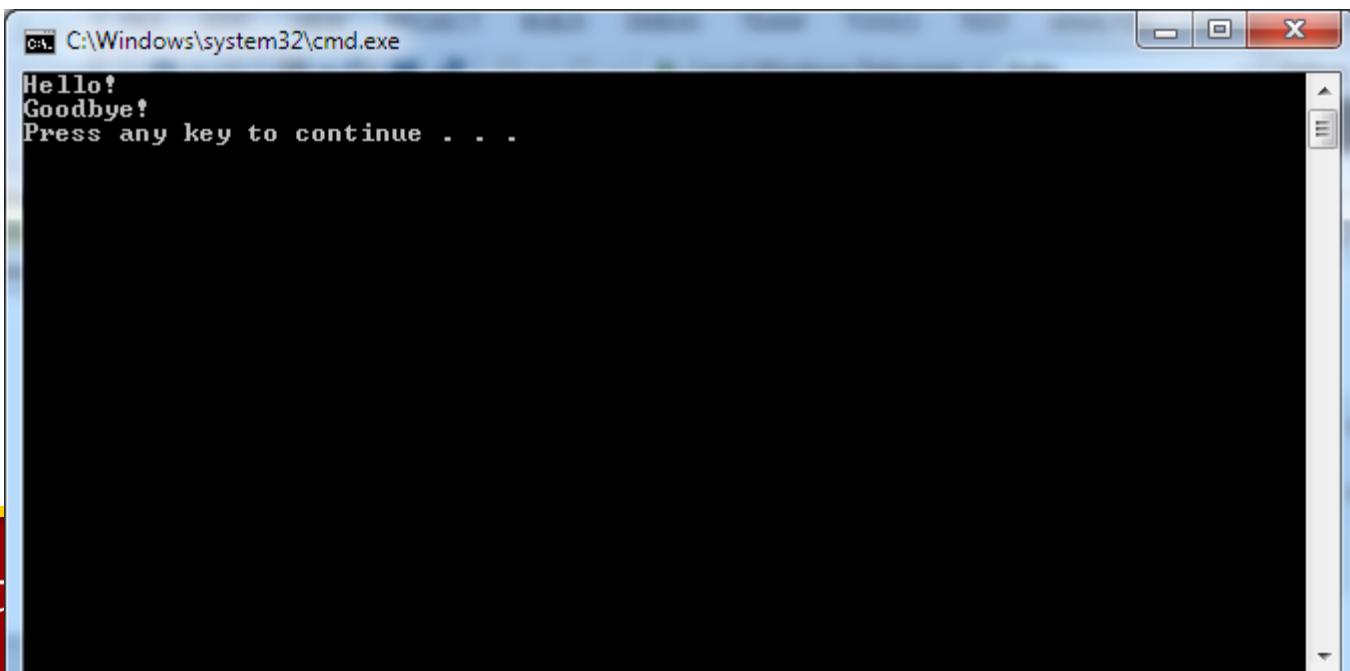
A screenshot of a Windows command prompt window titled 'cmd C:\Windows\system32\cmd.exe'. The window displays the text 'Hello!Goodbye!Press any key to continue . . .'.

```
cmd C:\Windows\system32\cmd.exe
Hello!Goodbye!Press any key to continue . . .
```



# 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;
}
```