C++ Basics
Announcements

Lab 1 this week!
Syntax

Syntax is a fancy word for the “grammar” of programming languages.

The basic English syntax is:
(subject) (verb) (noun)
“I eat bananas” not “Bananas I eat”

The computer is VERY picky (and stubborn) about grammar, and will not understand you unless you are absolutely correct!
Braces denote a block of code \{ \} (belonging to a method, class, etc.)

“White space” is ignored, just as the your brain will ignore the bottom third of this slide (this is why we need a semi-colon)
Variables

Variables are objects in program

To use variables two things must be done:
- Declaration
- Initialization

See: uninitialized.cpp

Example:
I am 0 inches tall.
I am -1094369310 inches tall.
Variables

```c
int x, y, z;
// Declaration
x = 2;
// Initialization
y = 3;
z = 4;
```

Same as:

```c
int x=2, y=3, z=4;
```

Variables can be declared anywhere (preferably at start)
Assignment operator

= is the assignment operator

The object to the right of the equals sign is stored into the object in the left

```cpp
int x, int y;
y = 2;
x = y+2;
```

See: assignmentOp.cpp
Assignment operator

= is NOT a mathematic equals

x=3;
x=4;  // computer is happy!

This does not mean 3=4
Assignment operator

To the left of = needs to be a valid object that can store the type of data on the right

```java
int x;
x=2.6; // unhappy, 2.6 is not an integer

x+2 = 6; // x+2 not an object

2 = x; // 2 is a constant, cannot store x
```
Assignment operator

What does this code do?

```java
int x = 2, y = 3;
y=x;
x=y;
```

What was the intention of this code?
Increment operators

What does this code do?

```java
int x = 2;
x=x+1;
```

Same as:

```java
x+=1;  or
x++;  
```
Increment operators

Two types of increment operators:

`x++;`  // increments after command
`vs`
`++x;`  // increments before command
Complex assignments

The following format is general for common operations:

variable (operator)= expression
variable = variable (operator) expression

Examples:

\[ x += 2 \quad \text{↔} \quad x = x + 2 \]
\[ x *= y + 2 \quad \text{↔} \quad x = x * (y + 2) \]
Order of operations

Order of precedence (higher operations first):
- -, +, ++, -- and ! (unary operators)
* *, / and % (binary operators)
+ and - (binary operators)

% is remainder operator
(example later in simpleDivision.cpp)
Order of operations

Binary operators need two arguments
Examples: 2+3, 5/2 and 6%2

Unary operators require only one argument:
Examples: (see binaryVsUnaryOps.cpp) +x, x++, !x

(! is the logical inversion operator for bool)
Order of operations

When multiple operations have the same precedence level:

Binary operations go from left to right

Unary operations go right to left
Identifiers

Hello my name is
Inigo Montoya
You killed my Father
Prepare to die
An **identifier** is the name of a variable (or object, class, method, etc.)

- **Case sensitive**
- Must use only letters, numbers or _
- Cannot start with a number
- (Some reserved identifiers, like main)
Identifiers

Already did this in week 1! See: runtimeError.cpp

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

int main()
{
  int number;

  cout << "What is your lucky number?" << endl;
  cin >> number;
  cout << "I like " << number << "!\n";

  return 0;
}
```
Which identifiers are valid?

1) james parker
2) BoByBoY
3) x3
4) 3x
5) x________
6) ________x
7) Home.Class
8) Five%
9) x-1
Identifiers

Which identifiers are valid?

1) james parker
2) BoByBoY
3) x3
4) 3x
5) x________
6) ________x
7) Home.Class
8) Five%
9) x 1
Identifiers

(See: float.cpp)

```cpp
int main()
{
    float Float, fLoat, flOat, FLOAt, FLOAT;
    Float = 1;
    fLoat = 2;
    flOat = -3;
    FLOAt = 2;
    FLOAT = 4;
    cout << (-fLoat + floAT(fLoat*fLoat - FLOAt * Float * flOat))/(FLOAT*FLOAt);
    cout << (-fLoat - floAT(fLoat*fLoat - FLOAt * Float * flOat))/(FLOAT*FLOAt);
    return 0;
}
```
Identifiers