CSCI 1300

Intro to Computing

Gabe Johnson

Lecture 32 April 5, 2013

C++: More Syntax

Lecture Goals

1. Status of HW 7?
2. Inequalities, Equalities
3. Logical And, Or, Not
4. Casting Data Types
5. Size of Data Types
6. More Coding!

Upcoming Homework Assignment

HW #7 **Due: Friday, Apr 5**

CPP Basics

If you haven't started yet, *good luck*.

Today we're covering the rest of the C++ syntax you need to complete this assignment. We've already covered the *concepts* in Python and Java, but you haven't seen some of the C++ syntax for these concepts.

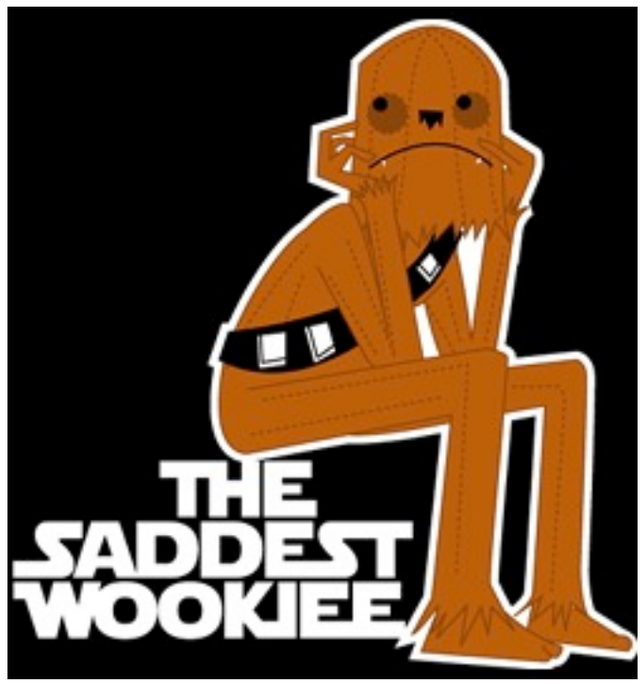
Status of HW 7

```
0: 40     | ****************************************
1: 0      |
2: 6      | ******
3: 0      |
4: 3      | ***
5: 0      |
6: 7      | *******
7: 0      |
8: 11     | ***********
9: 0      |
10: 77    | ***************************************************************************
Total students participating: 144
```

Projects: turn in summary

You may turn in your summary to RG by uploading a file that is named *exactly* summary.txt to HW7. Next week you'll do it to HW8, and so on.

While it gives you full points (so easy!), be aware that the group's TA will read them, and if you're uploading crap just to get the points, you will be a sad wookiee.



Inequalities

You can perform a true/false test to see how two numbers relate. One way of doing this is with an *inequality.* They look just like in Java and Python:

```
x < 4
x <= 4
x > 4
x >= 4
```

Inequalities

It might be tempting to use this syntax to see if x is in some range:

```
0 < x < 10
```

... however, while this is syntactically correct, it is not what you want. For example...

Inequalities

```
int x = -7;
cout << "x: " << x << endl;
cout << "0 < x: " << (0 < x) << endl;
cout << "x < 10: " << (x < 10) << endl;
cout << "(0 < x && x < 10): " << (0 < x && x < 10) << endl;
cout << "0 < x < 10: " << (0 < x < 10) << endl;
```

```
x: -7
0 < x: 0
x < 10: 1
(0 < x && x < 10): 0
0 < x < 10: 1
```

Not what you wanted!

Booleans are 1 and 0

The reason why this doesn't behave how you expect is because a boolean expression is resolved to a numeric 0 or 1. If something is false, that expression is turned into a 0, if it is true, it is turned into a 1.

```
// x is -7
(0 < x < 10)
```

The part "0 < x" is false, and becomes 0. Then the expression looks to the computer like:

```
0 < 10
```

And, since 0 is less than 10, the result is true (which is actually 1).

Equalities

You can tell if a number is *exactly* the same value as another thing by using the == operator. You can invert this (see if a number is *not exactly* the same value) with the != operator. Examples:

```
x == 4
x != 4
```

Equalities vs Assignment

These are two completely different things:

`x == 4` The first one is an equality test. It asks ‘is x equal to 4?’

`x = 4` The second one is an assignment. It demands ‘put the value 4 in the variable x’.

Logical And

We can combine these true/false tests using the same boolean arithmetic concepts we've had since Python. It has Java-like syntax.

The logical 'and' operator is &&. Example:

```
if (x >= 0 && x <= 10) { ... }
```

Logical Or

The logical ‘or’ is used the same way but with the operator ||. The vertical lines are the *pipe* symbol, which is usually just about the Enter key. Examples:

```
if (x != y || y != z) { ... }
else if (x == z || z == 4) { ... }
```

Logical Not

We can flip boolean statements around by using the logical not operator, which is a single exclamation point. Examples:

```
bool happy = (x > 0 && x < 10);
bool opposite = !happy;

bool g = is_green(x);
bool opposite_of_g = !is_green(x);
```

Casting Data Types

In Python if we wanted to convert a variable to some other type like an int, we wrote:

```
x_as_int = int(x)
```

In C++ the idea is the same, but the syntax is not:

```
int x = (float) x;
```

Size of Data Types

If you want to find the size of some given data type (like in for the ‘size of n doubles’ question in HW7), you can use the ‘sizeof’ function. Two ways to do this---using a variable, or using a data type.

```
double x = 42.349;
size_t a = sizeof(x);
size_t b = sizeof(double);
```

size_t ?

```
size_t b = sizeof(double);
```

OK so what's this 'size_t' thing?

It is actually a data type that is commonly used to measure the size of things in memory. Since things can't have negative size, a size_t is *unsigned*, meaning it can hold no negative value.

You can treat this like an int. You might have to cast it to an int, but you know how to do that now.