# Lecture 4: CS2400 Introduction to Computer Science

- Arithmetic
- Conditionals
  - comparison operators
  - logical connectives

Note: `pu1.cs.ohio.edu` is available to `ssh` into and is identical to the ones in Stocker 307, but is in the server room, so will not be turned off.

Note: error in last lecture, $\pi \approx 3.14159265358979323846264338327950288419716939937510582097 49$

# Arithmetic Operators and Expressions

```
+  *  /  -
```

What happens when the operands are of the same type?

```
7.0/2.0 =
```

```
7/2 =
```

# Mixing Types in Arithmetic Expressions

When one operand is `double` and one is `int` then the result is of type `double`.

What is `7.0/4`?

Are `6.0/3` and `6/3` the same?

# The Mod Operator

The operator '`%`' is used to get the remainder in an integer division problem. For example if you divide 13 by 3 you get 4 with remainder 1.

How could you get 4 in C++ from 13 and 3?

How can we get 1?

There is also a built in operator to accomplish the same thing. It is called with the '`%`' character.

E.G.: `13 % 3`

# Warning!

Contrary to your expectations, `/` and `%` may give different values on different systems when used with negative values!!

# Parentheses

It is, in general, a good idea to put parentheses in any non-trivial arithmetic expression.

Why?

# What if there are no parentheses?

The computer uses precedence rules to determine what to combine first.

Examples:

```
b*b - 4 * a * c
```

```
speed * time_to_point_a + time_to_point_b
```

Write a C++ expression for the following math formula $\frac{a+b}{cd-bc}$

# Shorthand Statements

If you want to update the value of a particular variable by multiplying, dividing, adding, or subtracting a value from itself, then there is a shorthand way of doing it:

| Example: | Equivalent to: |
|---|---|
| `count+= 3;` | `count = count + 3;` |
| `total-=discount*price;` | `total = total - (discount * price);` |
| `bunnies*=4;` | `bunnies = bunnies * 4;` |
| `amoeba/=2;` | `amoeba = amoeba / 2;` |
| `cents%=100;` | `cents = cents % 100;` |
| `zoo+=tigers+bears+lions;` | `zoo = zoo + tigers + bears + lions;` |

# Flow of Control

The `if-else` statement is a way of changing what the program does depending on the result of a test.

E.G.

```
if (good < min_good){
   cout << "You get coal!\n";
} else {
   cout << "You've been good, you get candy!\n";
}
```

Only one of the `cout` statements will be executed. The comparison between `good` and `min_good` determines which statement will be executed.

# Formal Syntax of `if-else` statements:

```
if (Logical_Expression)
   Yes_Statement
else
   No_Statement
```

or:

```
if (Logical_Expression)
{
   Yes_Statement_1
   Yes_Statement_2
   ...
   Yes_Statement_Last
} else {
   No_Statement_1
   No_Statement_2
   ...
   No_Statement_Last
}
```

# Comparison Operators

| Math Symbol | C++ Symbol |
|---|---|
| = | == |
| ≠ | != |
| < | < |
| ≤ | <= |
| > | > |
| ≥ | >= |

# Logical Expressions

What if we want to test for multiple things being true? For example, what if we want a number to be in the range 0-10? How can we test for this condition?

```
(0 < choice < 10)
```

will this work?

# Logical Connectives

To connect together logical expressions we can use ***logical connectives***. These enable us to build up more complex logical tests from simple ones.

There are three basic connectives:

- `&&` logical and
- `||` logical or
- `!` logical not

What do these do?

# Examples:

- Is there an error in the following?

```
if ((x < y) < z)
  cout << "y is between x and z.\n";
else
  cout << "y is out of bounds.\n";
```

- Is there an error in the following?

```
if (x = 42)
  cout << "I have the answer!\n";
else
  cout << "Still Searching!\n";
```

What do you do if there is no `else` clause?

What do you do if there is no first clause?

# Style

Even if you initially do not have more than one statement for a clause of the if statement, it is still usually a good idea to use the compound format:

```
if (Logical_Expression)
{
   Yes_Statement_1
   Yes_Statement_2
   ...
   Yes_Statement_Last
} else {
   No_Statement_1
   No_Statement_2
   ...
   No_Statement_Last
}
```

# Escape Sequences

So far we have seen the '`\n`' character.

What does the `\` mean?

What other `\` values make sense?

| | |
|---|---|
| New line | `\n` |
| Horizontal tab | `\t` |
| Backslash | `\\` |
| Alert | `\a` |
| Double quote | `\"` |

# Looping

How can we repeat the same set of statements a number of times?