For next time, read LCR 4.

On the Mac, there is an Integrated Development System (IDE) called Xcode, which includes a C++ compiler. Part of the Developers Tools. You can get it off the OSX CD or download from apple.com.

The Eclipse system, which I use in class, is an IDE, which runs on everything (Mac, Windows, linux). It's free. (But I don't know if it includes a C++ compiler.)

Visual Studio Express (?) for Windows is free?
Code Blocks is free, includes C++?

<u>Values and Objects</u>:

C++ is called an "object-oriented programming language" (OOPL) because it's designed for "object-oriented programming" (OOP).
It will be clearer by the end of this class what OOP is, but for now, I'll say it was a new method of programming, developing around 1980 (but with roots stretching into the '60s), for developing and maintaining programs more reliably. Better software engineering. An OOPL is a language designed to better support OOP.

So in C++ we have <u>values</u> (data, numbers, strings) that we operate on (+, -, *, /) to produce other data.

And we also have <u>objects</u>, which belong to <u>classes</u>, and have <u>behaviors</u> and <u>properties</u>. You can think of them like physical objects, but they're all in the computer.

An example of an object is "robot", which belongs to the class "scribbler." (That's

analogous to saying "Fido" is an object that belongs to the class "dog.")
Just like dogs have characteristic behaviors and properties, so do scribblers (they can go forward, turn, stop, they have a name, etc.).

To program with objects, we think of them as things (or even agents) that obey commands. This is how we give the forward command to robot:

```
robot.forward (1);
```

Values: We have types of values (ints, doubles, strings, chars,…) and operators that work on them.

In C++ and most other languages, some of the operators are overloaded, which means the same operator has several meanings.
For example, "+" can be used to add ints or doubles (or any other numeric types).
2 + 2 or 1.5 + 2.7. In C++ we can add strings.
For example,

```
 string UT = "University of Tennessee";
 string UTK = UT + " at Knoxville";
```

After these two statements, UTK has the value
  "University of Tennessee at Knoxville"

So addition for strings means concatenation (gluing them together end to end).

The compiler knows what to do by looking at the types of the operands.
If they're two ints, do int addition.
If they're two doubles, do double addition,
If they're two strings, do string concatenation.

There are several way to repeatedly execute code (iterate, loop) in C++. Definite iteration is where we know in advance how many times we are going around the loop. The basic way of doing this in C++ is the for-loop.

```
for ( <initialization> ; <continuation> ; <incrementation> ) {
      <statement 1>
      <statement 2>
      ….
      <statement n>
}
```

Statements 1 through n are repeated some number of times. Example:

```
for (int step = 1; step <= 6; step++) {
      // some statements
}
```

Typically the <initialization> is the declaration and initialization of an integer variable, e.g.:

```
int step = 1
```

The <continuation> is a condition (a test) that tells the conditions under which the looping should continue. For example,

```
step <= 6
```

means "so long as 'step' is less than or equal to 6".

}

The <incrementation> says how to change the loop variable ("step" in this loop) after each repetition. In this case it is

```
step++
```

which is a C++ abbreviation for "step = step + 1" and means "add 1 to 'step'."
Putting it all together:

```
for (int step = 1; step <= 6; step++) {
      <statements>
}
```

The other principal kind of iteration is indefinite iteration.
In this case I don't know exactly how many times it is going to loop, because it loops until some condition is satisfied (or not satisfied).
The principal way to do this in C++ is with a while-loop.

```
while (<condition>) {
      <statements>
}
```

This repeats the statements so long as the condition is true.

```
int n = 1024;
while (n > 0) {
      n = n/2;
      cout << n << endl;
}
```

What would this code do if I declared n <u>double</u>?