Statements vs. Expressions

Until now, I've gone along using the terms *statement* and *expression*. These are fundamental terms in C++, so it's important to clarify them. In general, you recognize a statement by its terminating semicolon (;).

```
cout << i++ << " ";
```

A simple statement such as this is usually one line of a C++ program. But remember that a semicolon terminates a statement, so it's legal (though not especially recommended) to put two statements on a line:

```
cout << i << " ";  i++;
```

Fine, you say—a statement is (usually) one line of a C++ program, terminated with a semicolon. So, what's an expression? An expression usually produces a value (with a few notable exceptions). You terminate an expression to get a simple statement. Here's a sample list of expressions, along with descriptions of what value each produces:

```
x              // Produces value of x
12             // Produces 12
x + 12         // Produces x + 12
x == 33        // Test for equality: true or false
x = 33         // Assignment: produces value assigned
++num          // Produces value before incrementing
i = num++ + 2  // Complex expression; produces
               //   new value of i
```

Because these are expressions, any of these can be used as part of a larger expression, including assignment (=). The last three have *side effects*. x = 33 alters the value of x, and num++ alters the value of num. The last example changes the value of both num and i. Remember, any expression can be turned into a statement by using a semicolon (;).

```
num++;
```

The fact that any expression can be turned into a statement this way makes some strange statements possible. You could, for example, turn a literal constant into a statement, but such a statement would do exactly nothing.