Lecture 6:
Introduction to C++ Programming (cont)

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
April 6th, 2010

2.6 Arithmetic (cont.)

Common Programming Error 2.4

*Some programming languages use operators ** or ^ to represent exponentiation. C++ does not support these exponentiation operators; using them for exponentiation results in errors.*

2.6 Arithmetic (cont.)

Good Programming Practice 2.11

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.

2.6 Arithmetic (cont.)

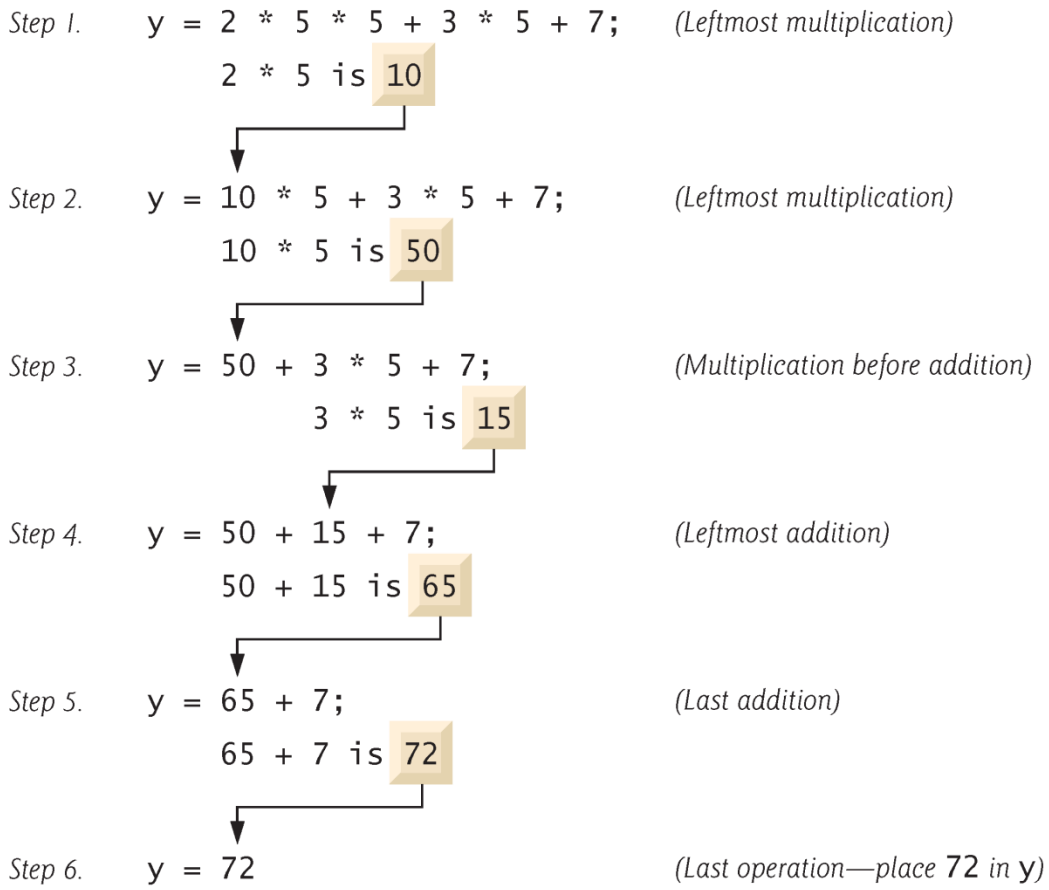

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

2.7 Decision Making: Equality and Relational Operators

- The if statement allows a program to take alternative action based on whether a condition is true or false.
- If the condition is true, the statement in the body of the `if` statement is executed.
- If the condition is false, the body statement is not executed.
- Conditions in `if` statements can be formed by using the equality operators and relational operators summarized in Fig. 2.12.
- The relational operators all have the same level of precedence and associate left to right.
- The equality operators both have the same level of precedence, which is lower than that of the relational operators, and associate left to right.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.5

A syntax error will occur if any of the operators ==, !=, >= and <= appears with spaces between its pair of symbols.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.6

Reversing the order of the pair of symbols in any of the operators !=, >= and <= (by writing them as =!, => and =<, respectively) is normally a syntax error. In some cases, writing != as =! will not be a syntax error, but almost certainly will be a ***logic error*** *that has an effect at execution time. You'll understand why when you learn about logical operators in Chapter 5. A* ***fatal logic error*** *causes a program to fail and terminate prematurely. A* ***nonfatal logic error*** *allows a program to continue executing, but usually produces incorrect results.*

2.7 Decision Making: Equality and Relational Operators

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Fig. 2.12 | Equality and relational operators.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.7

Confusing the equality operator == with the assignment operator = results in logic errors. The equality operator should be read "is equal to," and the assignment operator should be read "gets" or "gets the value of" or "is assigned the value of." Some people prefer to read the equality operator as "double equals." As we discuss in Section 5.9, confusing these operators may not necessarily cause an easy-to-recognize syntax error, but may cause extremely subtle logic errors.

2.7 Decision Making: Equality and Relational Operators (cont.)

- The following example uses six `if` statements to compare two numbers input by the user.
- If the condition in any of these `if` statements is satisfied, the output statement as-sociated with that `if` statement is executed.
- Figure 2.13 shows the program and the input/output dialogs of three sample executions.

2.7 Decision Making: Equality and Relational Operators

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // allows program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13    int number1; // first integer to compare
14    int number2; // second integer to compare
15
16    cout << "Enter two integers to compare: "; // prompt user for data
17    cin >> number1 >> number2; // read two integers from user
18
19    if ( number1 == number2 )
20       cout << number1 << " == " << number2 << endl;
21
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 1 of 3.)

2.7 Decision Making: Equality and Relational Operators

```
22      if ( number1 != number2 )
23         cout << number1 << " != " << number2 << endl;
24
25      if ( number1 < number2 )
26         cout << number1 << " < " << number2 << endl;
27
28      if ( number1 > number2 )
29         cout << number1 << " > " << number2 << endl;
30
31      if ( number1 <= number2 )
32         cout << number1 << " <= " << number2 << endl;
33
34      if ( number1 >= number2 )
35         cout << number1 << " >= " << number2 << endl;
36   } // end function main
```

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)

2.7 Decision Making: Equality and Relational Operators

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 3 of 3.)

2.7 Decision Making: Equality and Relational Operators (cont.)

- using declarations that eliminate the need to repeat the `std::` prefix as we did in earlier programs.
- Once we insert these `using` declarations, we can write `cout` instead of `std::cout`, `cin` instead of `std::cin` and `endl` instead of `std::endl`, respectively, in the remainder of the program.
- Many programmers prefer to use the declaration

```
using namespace std;
```

  which enables a program to use all the names in any standard C++ header file (such as `<iostream>`) that a program might include.
- From this point forward in the book, we'll use the preceding declaration in our programs.

2.7 Decision Making: Equality and Relational Operators (cont.)

- Each `if` statement in Fig. 2.13 has a single statement in its body and each body statement is indented.
- In Chapter 4 we show how to specify `if` statements with multiple-statement bodies (by enclosing the body statements in a pair of braces, `{ }`, creating what's called a compound statement or a block).

2.7 Decision Making: Equality and Relational Operators

Good Programming Practice 2.12

Indent the statement(s) in the body of an `if` *statement to enhance readability.*

2.7 Decision Making: Equality and Relational Operators

Good Programming Practice 2.13

For readability, there should be no more than one statement per line in a program.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.8

Placing a semicolon immediately after the right parenthesis after the condition in an `if` *statement is often a logic error (although not a syntax error). The semicolon causes the body of the* `if` *statement to be empty, so the* `if` *statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the* `if` *statement now becomes a statement in sequence with the* `if` *statement and always executes, often causing the program to produce incorrect results.*

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.9

It's a syntax error to split an identifier by inserting white-space characters (e.g., writing `main` *as* `ma in`*).*

2.7 Decision Making: Equality and Relational Operators

Good Programming Practice 2.14

A lengthy statement may be spread over several lines. If a single statement must be split across lines, choose meaningful breaking points, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines and left-align the group of indented lines.

2.7 Decision Making: Equality and Relational Operators (cont.)

- Figure 2.14 shows the precedence and associativity of the operators introduced in this chapter.
- The operators are shown top to bottom in decreasing order of precedence.
- All these operators, with the exception of the assignment operator =, associate from left to right.

2.7 Decision Making: Equality and Relational Operators

Operators				Associativity	Type
()				left to right	parentheses
*	/	%		left to right	multiplicative
+	-			left to right	additive
<<	>>			left to right	stream insertion/extraction
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

Questions