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
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.
Good Programming Practice 2.11

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.
2.6 Arithmetic (cont.)

Step 1. \( y = 2 \times 5 \times 5 + 3 \times 5 + 7; \)  
\( 2 \times 5 \) is \( 10 \)

Step 2. \( y = 10 \times 5 + 3 \times 5 + 7; \)  
\( 10 \times 5 \) is \( 50 \)

Step 3. \( y = 50 + 3 \times 5 + 7; \)  
\( 3 \times 5 \) is \( 15 \)

Step 4. \( y = 50 + 15 + 7; \)  
\( 50 + 15 \) is \( 65 \)

Step 5. \( y = 65 + 7; \)  
\( 65 + 7 \) is \( 72 \)

Step 6. \( y = 72 \)  
(Last operation—place 72 in \( y \))

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.
• 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.
Common Programming Error 2.5

A syntax error will occur if any of the operators `==`, `!=`, `>=` and `<=` appears with spaces between its pair of symbols.
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 non-fatal logic error allows a program to continue executing, but usually produces incorrect results.
## 2.7 Decision Making: Equality and Relational Operators

<table>
<thead>
<tr>
<th>Standard algebraic equality or relational operator</th>
<th>C++ equality or relational operator</th>
<th>Sample C++ condition</th>
<th>Meaning of C++ condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Relational operators</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>≥</td>
<td>&gt;=</td>
<td>x &gt;= y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>≤</td>
<td>&lt;=</td>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
<tr>
<td><strong>Equality operators</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>≠</td>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
</tbody>
</table>

**Fig. 2.12**  | 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.
• 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 associated 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

// Fig. 2.13: fig02_13.cpp
// Comparing integers using if statements, relational operators
// and equality operators.
#include <iostream> // allows program to perform input and output
using std::cout; // program uses cout
using std::cin; // program uses cin
using std::endl; // program uses endl

// function main begins program execution
int main()
{
    int number1; // first integer to compare
    int number2; // second integer to compare

    cout << "Enter two integers to compare: "; // prompt user for data
    cin >> number1 >> number2; // read two integers from user

    if ( number1 == number2 )
        cout << number1 << " == " << number2 << endl;

Fig. 2.13   Comparing integers using if statements, relational operators and equality operators. (Part I of 3.)
2.7 Decision Making: Equality and Relational Operators

```cpp
if ( number1 != number2 )
    cout << number1 << " != " << number2 << endl;
if ( number1 < number2 )
    cout << number1 << " < " << number2 << endl;
if ( number1 > number2 )
    cout << number1 << " > " << number2 << endl;
if ( number1 <= number2 )
    cout << number1 << " <= " << number2 << endl;
if ( number1 >= number2 )
    cout << number1 << " >= " << number2 << endl;
```

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: \(22\ 12\)
- \(22 \neq 12\)
- \(22 > 12\)
- \(22 \geq 12\)

Enter two integers to compare: \(7\ 7\)
- \(7 == 7\)
- \(7 \leq 7\)
- \(7 \geq 7\)

**Fig. 2.13** Comparing integers using if statements, relational operators and equality operators. (Part 3 of 3.)
• **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.
• 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.
Good Programming Practice 2.13

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

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( )</td>
<td>left to right</td>
<td>parentheses</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>multiplicative</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td>additive</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>left to right</td>
<td>stream insertion/extraction</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td>== !=</td>
<td>left to right</td>
<td>equality</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
</tbody>
</table>

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