Introduction to C++ Programming, Input/ Output and Operators

What's in a name? that
which we call a rose
By any other name
would smell as sweet.

—**William Shakespeare**

High thoughts must have high language.

—**Aristophanes**

One person can make a difference and every person should try.

—**John F. Kennedy**

OBJECTIVES

In this chapter you'll learn:

- To write simple computer programs in C++.
- To write simple input and output statements.
- To use fundamental types.
- Basic computer memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write simple decision-making statements.

2.1 Introduction
2.2 First Program in C++: Printing a Line of Text
2.3 Modifying Our First C++ Program
2.4 Another C++ Program: Adding Integers
2.5 Memory Concepts
2.6 Arithmetic
2.7 Decision Making: Equality and Relational Operators
2.8 Wrap-Up

2.1 Introduction

- We now introduce C++ programming, which facilitates a disciplined approach to program development.
- Most of the C++ programs you'll study in this book process data and display results.

2.2 First Program in C++: Printing a Line of Text

- Simple program that prints a line of text (Fig. 2.1).

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

2.2 First Program in C++: Printing a Line of Text (cont.)

- // indicates that the remainder of each line is a comment.
  - You insert comments to document your programs and to help other people read and understand them.
  - Comments are ignored by the C++ compiler and do not cause any machine-language object code to be generated.
- A comment beginning with // is called a single-line comment because it terminates at the end of the current line.
- You also may use comments containing one or more lines enclosed in /* and */.

Good Programming Practice 2.1

Every program should begin with a comment that describes the purpose of the program.

2.2 First Program in C++: Printing a Line of Text (cont.)

- A preprocessing directive is a message to the C++ preprocessor.
- Lines that begin with # are processed by the preprocessor before the program is compiled.
- `#include <iostream>` notifies the preprocessor to include in the program the contents of the input/output stream header file <iostream>.
  - This header is a file containing information used by the compiler when compiling any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.

Common Programming Error 2.1

Forgetting to include the `<iostream>` header in a program that inputs data from the keyboard or outputs data to the screen causes the compiler to issue an error message.

2.2 First Program in C++: Printing a Line of Text (cont.)

- You use blank lines, *space characters* and *tab characters* (i.e., "tabs") to make programs easier to read.
  - Together, these characters are known as white space.
  - White-space characters are normally *ignored* by the compiler.

2.2 First Program in C++: Printing a Line of Text (cont.)

- `main` is a part of every C++ program.
- The parentheses after `main` indicate that main is a program building block called a function.
- C++ programs typically consist of one or more functions and classes.
- Exactly *one* function in every program *must* be named `main`.
- C++ programs begin executing at function `main`, even if `main` is *not* the first function defined in the program.
- The keyword `int` to the left of `main` indicates that `main` "returns" an integer (whole number) value.
  - A keyword is a word in code that is reserved by C++ for a specific use.
  - For now, simply include the keyword `int` to the left of `main` in each of your programs.

2.2 First Program in C++: Printing a Line of Text (cont.)

- A left brace, {, must *begin* the body of every function.
- A corresponding right brace, }, must *end* each function's body.
- A statement instructs the computer to perform an action.
- Together, the quotation marks and the characters between them are called a string, a character string or a string literal.
- We refer to characters between double quotation marks simply as strings.
  - White-space characters in strings are not ignored by the compiler.
- Most C++ statements end with a semicolon (;), also known as the statement terminator.
  - Preprocessing directives (like `#include`) do not end with a semicolon.

Common Programming Error 2.2

Omitting the semicolon at the end of a C++ statement is a syntax error. The syntax of a programming language specifies the rules for creating proper programs in that language. A syntax error occurs when the compiler encounters code that violates C++'s language rules (i.e., its syntax). The compiler normally issues an error message to help you locate and fix the incorrect code. Syntax errors are also called compiler errors, compile-time errors or compilation errors, because the compiler detects them during the compilation phase. You cannot execute your program until you correct all the syntax errors in it. As you'll see, some compilation errors are not syntax errors.

Good Programming Practice 2.2

Indent the body of each function one level within the braces that delimit the function's body. This makes a program's functional structure stand out and makes the program easier to read.

Good Programming Practice 2.3

Set a convention for the size of indent you prefer, then apply it uniformly. The tab key may be used to create indents, but tab stops may vary. We prefer three spaces per level of indent.

2.2 First Program in C++: Printing a Line of Text (cont.)

- Typically, output and input in C++ are accomplished with streams of characters.
- When a `cout` statement executes, it sends a stream of characters to the standard output stream object—std::cout—which is normally "connected" to the screen.
- The `std::` before `cout` is required when we use names that we've brought into the program by the preprocessing directive `#include <iostream>`.
  - The notation `std::cout` specifies that we are using a name, in this case `cout`, that belongs to "namespace" `std`.
  - The names `cin` (the standard input stream) and `cerr` (the standard error stream) also belong to namespace `std`.

2.2 First Program in C++: Printing a Line of Text (cont.)

- In the context of an output statement, the << operator is referred to as the stream insertion operator.
  - The value to the operator's right, the right operand, is inserted in the output stream.
- The characters \n are *not* printed on the screen.
- The backslash (\) is called an escape character.
  - It indicates that a "special" character is to be output.
- When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an escape sequence.
- The escape sequence \n means newline.
  - Causes the cursor to move to the beginning of the next line on the screen.

Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\'`	Single quote. Used to print a single quote character.
`\"`	Double quote. Used to print a double quote character.

Fig. 2.2 | Escape sequences.

2.2 First Program in C++: Printing a Line of Text (cont.)

- When the return statement is used at the end of `main` the value `0` indicates that the program has *terminated successfully*.
- According to the C++ standard, if program execution reaches the end of `main` without encountering a `return` statement, it's assumed that the program terminated successfully—exactly as when the last statement in main is a `return` statement with the value `0`.

2.3 Modifying Our First C++ Program

- `Welcome to C++!` can be printed several ways.

```
1  // Fig. 2.3: fig02_03.cpp
2  // Printing a line of text with multiple statements.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome ";
9     std::cout << "to C++!\n";
10 } // end function main
```

```
Welcome to C++!
```

Fig. 2.3 | Printing a line of text with multiple statements.

2.3 Modifying Our First C++ Program (cont.)

- A single statement can print multiple lines by using newline characters.
- Each time the \n (newline) escape sequence is encountered in the output stream, the screen cursor is positioned to the beginning of the next line.
- To get a blank line in your output, place two newline characters back to back.

```
1  // Fig. 2.4: fig02_04.cpp
2  // Printing multiple lines of text with a single statement.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome\nto\n\nC++!\n";
9  } // end function main
```

```
Welcome
to

C++!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

2.4 Another C++ Program: Adding Integers

- The next program obtains two integers typed by a user at the keyboard, computes the sum of these values and outputs the result using `std::cout`.
- Figure 2.5 shows the program and sample inputs and outputs.

```
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two integers.
3  #include <iostream> // allows program to perform input and output
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1 = 0; // first integer to add (initialized to 0)
10    int number2 = 0; // second integer to add (initialized to 0)
11    int sum = 0; // sum of number1 and number2 (initialized to 0)
12
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
22 } // end function main
```

Fig. 2.5 | Addition program that displays the sum of two integers.

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 2.5 | Addition program that displays the sum of two integers.

2.4 Another C++ Program: Adding Integers (cont.)

- Declarations introduce identifiers into programs.
- The identifiers `number1`, `number2` and `sum` are the names of variables.
- A variable is a location in the computer's memory where a value can be stored for use by a program.
- Variables `number1`, `number2` and `sum` are data of type int, meaning that these variables will hold integer values, i.e., whole numbers such as 7, –11, 0 and 31914.
- All variables *must* be declared with a *name* and a *data type* before they can be used in a program.
- If more than one name is declared in a declaration (as shown here), the names are separated by commas (,); this is referred to as a comma-separated list.

Error-Prevention Tip 2.1

Although it's not always necessary to initialize every variable explicitly, doing so will help you avoid many kinds of problems.

Good Programming Practice 2.4

Declare only one variable in each declaration and provide a comment that explains the variable's purpose in the program.

2.4 Another C++ Program: Adding Integers (cont.)

- Data type `double` is for specifying real numbers, and data type `char` for specifying *character data*.
- Real numbers are numbers with decimal points, such as 3.4, 0.0 and –11.19.
- A `char` variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., $ or *).
- Types such as `int`, `double` and `char` are called fundamental types.
- Fundamental-type names are keywords and therefore *must* appear in all lowercase letters.
- Appendix C contains the complete list of fundamental types.

2.4 Another C++ Program: Adding Integers (cont.)

- A variable name is any valid identifier that is *not* a keyword.
- An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.
- C++ is case sensitive—uppercase and lowercase letters are different, so `a1` and `A1` are *different* identifiers.

Portability Tip 2.1

C++ allows identifiers of any length, but your C++ implementation may restrict identifier lengths. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice 2.5

Choosing meaningful identifiers makes a program self-documenting—a person can understand the program simply by reading it rather than having to refer to program comments or documentation.

Good Programming Practice 2.6

Avoid using abbreviations in identifiers. This improves program readability.

Good Programming Practice 2.7

Do not use identifiers that begin with underscores and double underscores, because C++ compilers may use names like that for their own purposes internally. This will prevent the names you choose from being confused with names the compilers choose.

2.4 Another C++ Program: Adding Integers (cont.)

- Declarations of variables can be placed almost anywhere in a program, but they must appear *before* their corresponding variables are used in the program.

2.4 Another C++ Program: Adding Integers (cont.)

- A prompt it directs the user to take a specific action.
- A `cin` statement uses the input stream object `cin` (of namespace `std`) and the stream extraction operator, `>>`, to obtain a value from the keyboard.
- Using the stream extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard.

2.4 Another C++ Program: Adding Integers (cont.)

- When the computer executes an input statement that places a value in an `int` variable, it waits for the user to enter a value for variable `number1`.
- The user responds by typing the number (as characters) then pressing the *Enter* key (sometimes called the Return key) to send the characters to the computer.
- The computer converts the character representation of the number to an integer and assigns (i.e., copies) this number (or value) to the variable `number1`.
- Any subsequent references to `number1` in this program will use this same value.

2.4 Another C++ Program: Adding Integers (cont.)

- In this program, an assignment statement adds the values of variables `number1` and `number2` and assigns the result to variable `sum` using the assignment operator `=`.
  - Most calculations are performed in assignment statements.
- The = operator and the + operator are called binary operators because each has two operands.

Good Programming Practice 2.8

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

2.4 Another C++ Program: Adding Integers (cont.)

- `std::endl` is a so-called stream manipulator.
- The name `endl` is an abbreviation for "end line" and belongs to namespace `std`.
- The `std::endl` stream manipulator outputs a newline, then "flushes the output buffer."
  - This simply means that, on some systems where outputs accumulate in the machine until there are enough to "make it worthwhile" to display them on the screen, `std::endl` forces any accumulated outputs to be displayed at that moment.
  - This can be important when the outputs are prompting the user for an action, such as entering data.

2.4 Another C++ Program: Adding Integers (cont.)

- Using multiple stream insertion operators (<<) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations.
- Calculations can also be performed in output statements.

2.5 Memory Concepts

- Variable names such as `number1`, `number2` and `sum` actually correspond to locations in the computer's memory.
- Every variable has a name, a type, a size and a value.
- When a value is placed in a memory location, the value overwrites the previous value in that location; thus, placing a new value into a memory location is said to be destructive.
- When a value is read out of a memory loca-tion, the process is nondestructive.

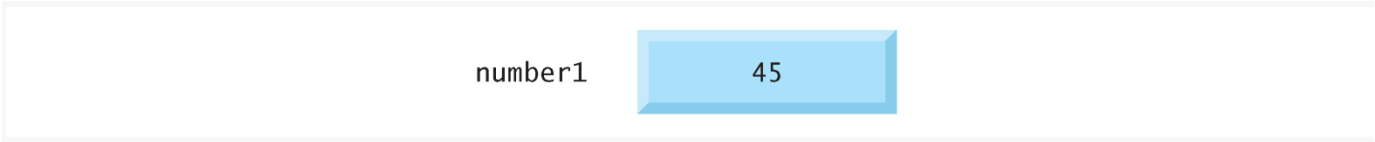

Fig. 2.6 | Memory location showing the name and value of variable `number1`.

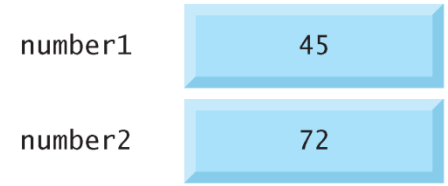

Fig. 2.7 | Memory locations after storing values in the variables for `number1` and `number2`.

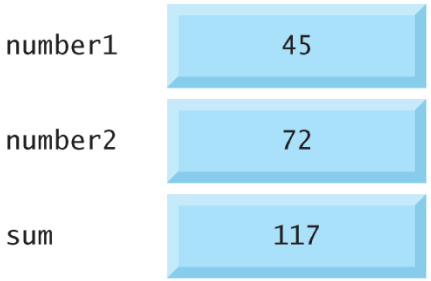

Fig. 2.8 | Memory locations after calculating and storing the `sum` of `number1` and `number2`.

2.6 Arithmetic

- Most programs perform arithmetic calculations.
- Figure 2.9 summarizes the C++ arithmetic operators.
- The asterisk (*) indicates multiplication.
- The percent sign (%) is the modulus operator that will be discussed shortly.
  - C++ provides the modulus operator, %, that yields the remainder after integer division.
  - The modulus operator can be used only with integer operands.
- The arithmetic operators in Fig. 2.9 are all binary operators.
- Integer division (i.e., where both the numerator and the denominator are integers) yields an integer quotient.
  - Any fractional part in integer division is discarded (i.e., truncated)—no rounding occurs.

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Fig. 2.9 | Arithmetic operators.

2.6 Arithmetic (cont.)

- Arithmetic expressions in C++ must be entered into the computer in straight-line form.
- Expressions such as "a divided by b" must be written as `a / b`, so that all constants, variables and operators appear in a straight line.
- Parentheses are used in C++ expressions in the same manner as in algebraic expressions.
- For example, to multiply `a` times the quantity `b + c` we write `a a * ( b + c )`.

2.6 Arithmetic (cont.)

- C++ applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those followed in algebra.

Operator(s)	Operation(s)	Order of evaluation (precedence)
`( )`	Parentheses	Evaluated first. If the parentheses are *nested*, such as in the expression `a * ( b + c / d + e )`, the expression in the *innermost* pair is evaluated first. [*Caution:* If you have an expression such as `(a + b) * (c - d)` in which two sets of parentheses are not nested, but appear "on the same level," the C++ Standard does *not* specify the order in which these parenthesized subexpressions will be evaluated.]

`*`
`/`
`%`
	Multiplication
	Division
	Modulus
		Evaluated second. If there are several, they're evaluated left to right.

`+` `-`	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Fig. 2.10 | Precedence of arithmetic operators.

2.6 Arithmetic (cont.)

- There is no arithmetic operator for exponentiation in C ++, so x^2 is represented as x * x.
- Figure 2.11 illustrates the order in which the operators in a second-degree polynomial are applied.
- As in algebra, it's acceptable to place unnecessary parentheses in an expression to make the expression clearer.
- These are called redundant parentheses.

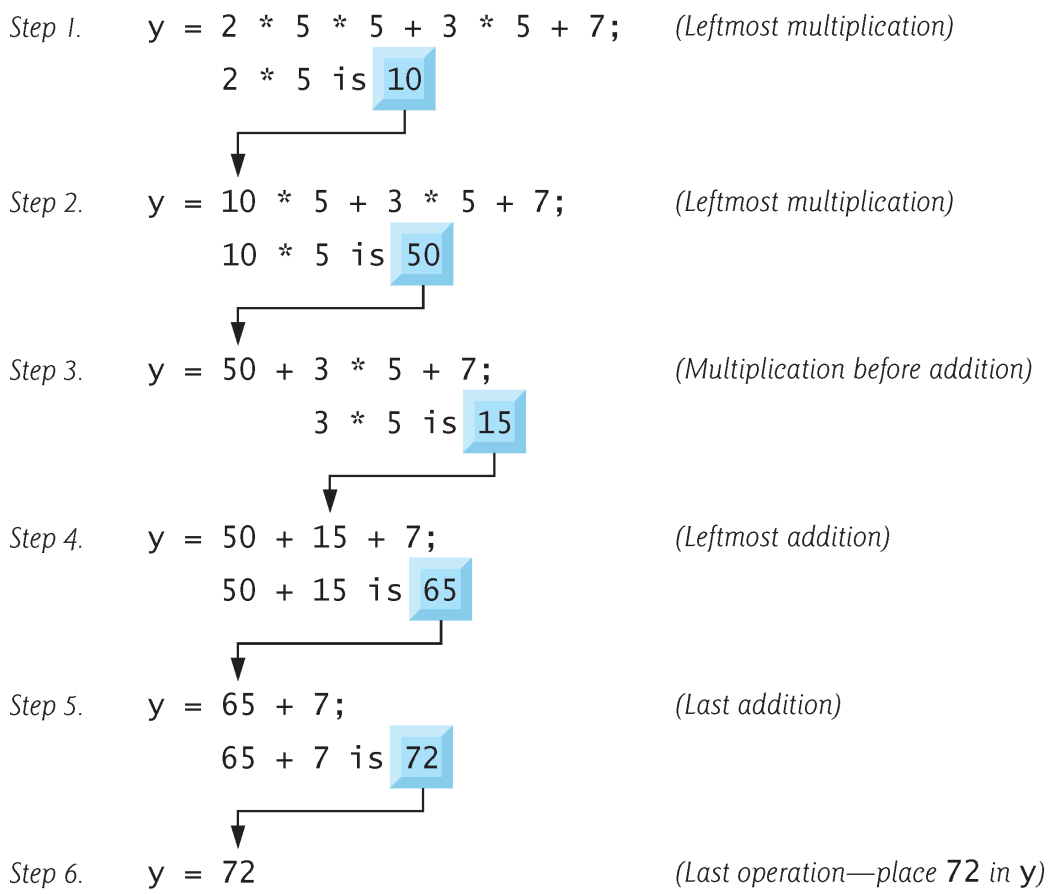

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

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.

Algebraic relational or equality operator	C++ relational or equality 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 | Relational and equality operators.

Common Programming Error 2.3

Reversing the order of the pair of symbols in 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.

Common Programming Error 2.4

Confusing the equality operator == with the assignment operator = results in logic errors. We like to read the equality operator as "is equal to" or "double equals," and the assignment operator as "gets" or "gets the value of" or "is assigned the value of." As you'll see in Section 5.9, confusing these operators may not necessarily cause an easy-to-recognize syntax error, but may cause 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.

```
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 = 0; // first integer to compare (initialized to 0)
14    int number2 = 0; // second integer to compare (initialized to 0)
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.)

```
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.)

```
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).

Good Programming Practice 2.9

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

Common Programming Error 2.5

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 (cont.)

- Statements may be split over several lines and may be spaced according to your prefer-ences.
- It's a syntax error to split identifiers, strings (such as `"hello"`) and constants (such as the number `1000`) over several lines.

Good Programming Practice 2.10

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.

Operators				Associativity	Type
()				*[See caution in Fig. 2.10]*	grouping 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.

Good Programming Practice 2.11

Refer to the operator precedence and associativity chart (Appendix A) when writing expressions containing many operators. Confirm that the operators in the expression are performed in the order you expect. If you're uncertain about the order of evaluation in a complex expression, break the expression into smaller statements or use parentheses to force the order of evaluation, exactly as you'd do in an algebraic expression. Be sure to observe that some operators such as assignment (=) associate right to left rather than left to right.