009992: Problem Solving and Lab: C++

Introduction to C++ Programming I/O and Operators

OBJECTIVES

- Simple computer programs in C++
- I/O Statements
- Data types
- Arithmetic and relational operators and precedence
- Simple decision making statements

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

```
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.)

- 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.

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.)

- 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.

2.3 Modifying Our First C++ Program

```
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.

```
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

```
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.)

- 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.

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.

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.)

- `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.

2.6 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.

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.

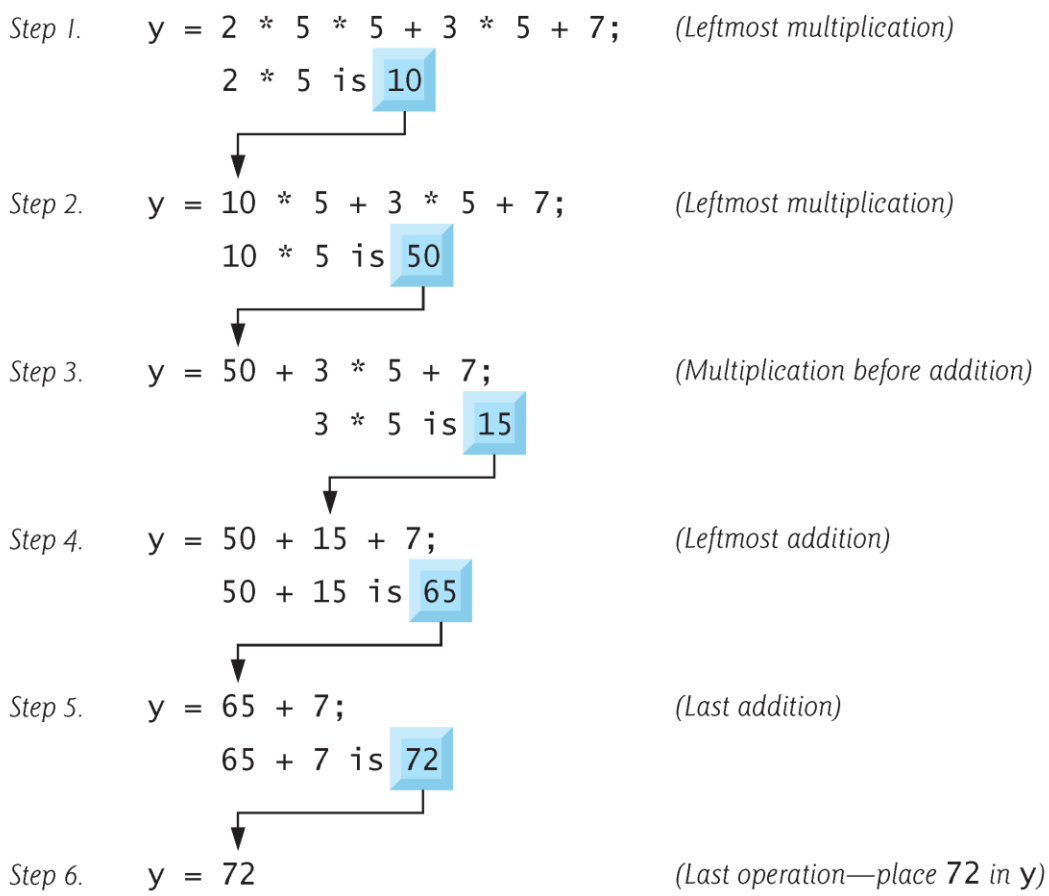

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

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.

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 course, 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 later tutorial 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 (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.

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

- Figure 2.14 shows the precedence and associativity of the operators introduced in this lesson.
- 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.