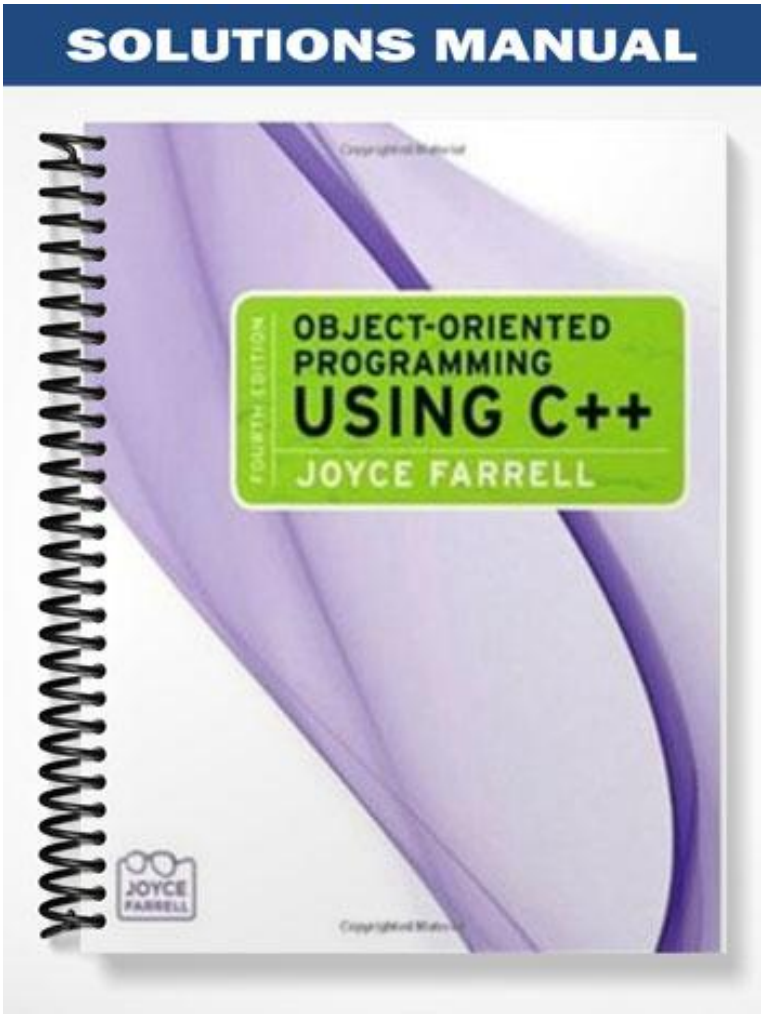
SOLUTIONS MANUAL



Chapter 2

Evaluating C++ Expressions

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

In Chapter 2, students learn how C++ evaluates expressions. They learn about the different binary and unary operators. Students learn about the precedence and associativity of arithmetic operations and examine shortcut arithmetic operators. They also learn to evaluate Boolean expressions and how to perform operations on structure fields.

Objectives

- Use C++ binary arithmetic operators
- Learn about the precedence and associativity of arithmetic operations
- Examine shortcut arithmetic operators
- Use other unary operators
- Evaluate Boolean expressions
- Perform arithmetic with class fields

Teaching Tips

Using C++ Binary Arithmetic Operators

1. Briefly introduce each of the five simple arithmetic operators provided by C++, and note that they are binary operators.

2. Stress that some arithmetic operators are not binary and some binary operators are not arithmetic.

3. Note that the results of an arithmetic operation can be used immediately or stored in a variable in computer memory (see Figure 2-1).

Teaching Tip	Explain that various programming choices may make your code run faster or slower. Refer to the first tip on page 53 for an example.

4. Explain that addition, subtraction, multiplication, division, or modulus of any two integers results in an integer.

Teaching Tip	Integer division is usually confusing for beginning students and is a common source of programming errors. Make sure students understand why 10 / 8 is 0.

5. Note that if either or both of the operands in addition, subtraction, multiplication, or division is a floating-point number –that is, at least one operand is a float or a double—then the result is also a floating-point number.

6. Introduce the terms *mixed expression* and *unifying type*.

7. Describe the order of precedence of unifying types in binary arithmetic expressions. You may refer to Figures 2-3 and 2-4 to help clarify how the precedence of unifying types affects how binary arithmetic expressions are evaluated. Don't forget to explain what a *cast* is. Note the difference between an implicit cast and an explicit cast.

Teaching Tip	Refer to the ASCII table available at www.asciitable.com/ to explain that you can convert a character to its integer equivalent using a cast, and vice versa.

Teaching Tip	Students may wonder if there is a difference between a C++ style static cast and a C style static cast. Although they may seem equivalent, there are some subtle differences between these types of casts. For more information, read: http://cboard.cprogramming.com/showthread.php?p=621213.

Using Modulus

1. Explain the use of the modulus operator (%) and note how it can be used to extract digits at the end of a number.

2. Introduce the concept of *check digits* and describe a simple algorithm for detecting and comparing check digits.

Quick Quiz 1

1. What is a mixed expression?
   Answer: An expression such as 3.2 * 2 is a mixed expression—one in which the operands have different data types.

2. What is an implicit cast?
   Answer: The automatic cast that occurs when you assign a value of one type to a type with higher precedence is called an implicit cast—meaning the cast is performed automatically and without your intervention.

3. The _____ gives the remainder of integer division; it can be used only with integers.

Answer: modulus operator (%)
modulus operator
% operator

4. A(n) _____ is a digit added to a number (either at the end or at the beginning) that validates the authenticity of the number.
Answer: check digit

Precedence and Associativity of Arithmetic Operators

1. Explain the concepts of *arithmetic precedence* and *associativity*.

2. Describe the steps that occur when C++ evaluates a mixed arithmetic expression.

3. Use Table 2-1 to help students practice the arithmetic expression rules they just learned.

4. Note how parentheses can be used to override precedence rules.

Shortcut Arithmetic Operators

1. Briefly introduce the two categories of shortcut arithmetic: compound and increment/decrement.

Compound Assignment Operators

1. Explain how compound assignment operators work.

Teaching Tip	A common beginner's mistake is to invert the order of compound assignment operators. Note that the operators +=, –=, *=, /=, and %= are all valid; the operators =+, =–, =*, =/, and =% are not. The assignment operator (=) always appears second in a compound assignment operator.

Increment and Decrement Operators

1. Introduce C++ increment and decrement operators. Make sure students understand the subtle difference between the prefix and postfix increment/decrement operators. Note that these operators are unary operators.

Quick Quiz 2

1. In C++, most operators have _____ associativity.
Answer: left-to-right

2. What is associativity?
   Answer: When two operations with the same precedence appear in an arithmetic expression, the operations are carried out in order from either left to right or right to left based on their associativity—the rule that dictates the order in which an operator works with its operands.

3. The add and assign operator is an example of a(n) _____—an operator that performs two tasks, one of which is assignment.
   Answer: compound assignment operator

4. _____ are those that require only one operand—that is, they perform an operation on only one side of the operator.
   Answer: Unary operators

Other Unary Operators

1. Note that besides the unary prefix and postfix increment and decrement operators, C++ supports several other unary operators, including +, –, and &.

2. Explain how the positive value and negative value operators work.

3. Explain how the address operator (&) works. Refer to Figures 2-6 and 2-7 to help explain the use of this operator.

Evaluating Boolean Expressions

1. Briefly explain what *relational operators* and *Boolean expressions* are. Table 2-2 lists the relational operators available in C++.

2. The *not* operator can be a bit confusing at first. Take some time to explain how it works with different expressions.

3. Stress that one should be very careful not to use = instead of == to compare two expressions. Using the assignment operator can lead to unexpected results that can be very hard to catch.

Performing Operations on `struct` Fields

1. Use Figures 2-8 through 2-10 to explain how to perform operations on `struct` fields.

Teaching Tip	At this point you may briefly note that when you create a class instead of a structure, some fields, called static fields, can be used without creating an object.

Quick Quiz 3

1. What is the positive value operator?
   Answer: When used alone in front of an operand, it becomes the positive value operator, indicating a positive value—one that is greater than zero.

2. _____ are those that evaluate the relationship between operands; you use these relational operators to evaluate Boolean expressions.
   Answer: Relational operators

3. A(n) _____ expression is one that is interpreted to be true or false.
   Answer: Boolean

4. What is the not operator?
   Answer: The unary operator ! is the not operator; it means "the opposite of," and essentially reverses the true/false value of an expression.

You Do It

1. Guide students as they work through the program in this section.

Using Arithmetic Operators

1. Be prepared to explain why students may need to use `getch()` in this program.

Teaching Tip	For more information on why `getch()` is needed and how to use it, see Appendix B and or www.daniweb.com/forums/thread11811.html.

Using Prefix and Postfix Increment and Decrement Operators

1. Make sure students have a good understanding of the difference between the prefix and the postfix increment/decrement operators.

Using Operators with `struct` Fields

1. Be prepared to solve any compilation problems that may arise from entering C++ code with syntax errors.

Class Discussion Topics

1. Do students have previous programming experience in other languages? If so, do they know of operators in other programming languages that do not exist in C++?

2. The use of the prefix and postfix increment/decrement operators can lead to confusing code. Some programmers believe that it is better not to use them unless the outcome is very clear and is guaranteed to avoid interpretation errors. Other programmers like to use these operators (and other similar C++ features) as much as possible, to make the code more concise. Ask students to give their opinions on this issue.

Additional Projects

1. C++ has other operators besides the ones introduced in this chapter. Ask students to research on the Internet for other operators and to compile a list of them (in order of precedence).

2. Ask students to find of at least two situations in which check digits are used, other than the ones covered in the book (note that some uses of check digits are also described in the Exercises section). For example, students may find out how check digits are used in ISBN (www.cs.queensu.ca/home/bradbury/checkdigit/isbncheck.htm) and UPC (www.cs.queensu.ca/home/bradbury/checkdigit/upccheck.htm) numbers.

Additional Resources

1. Operator Precedence in C++:
   http://cplus.about.com/od/learning1/ss/cppexpressionsr_7.htm

2. C++ Operator Precedence:
   www.cppreference.com/operator_precedence.html

3. Operators in C and C++:
   http://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

Key Terms

- The relational operator **!=** means not equal to.
- The relational operator < means less than.
- The relational operator <= means less than or equal to.
- The relational operator = = means equivalent to.
- The relational operator > means greater than.
- The relational operator >= means greater than or equal to.

- The **add and assign operator**, +=, adds the right-hand operand to the left-hand operand.
- The **addition operator** is the + symbol; it is a binary arithmetic operator used to perform addition.
- The **address operator** (&) is a unary operator used to refer to the memory address of a variable.
- An **arithmetic operator** is a symbol that performs arithmetic.
- **Arithmetic precedence** is the set of rules of order of performance of arithmetic operations. Operators with higher precedence are performed first in an arithmetic statement with multiple operations.
- **Associativity** is the rule that dictates the order in which an operator works with its operands.
- A **binary operator** is an operator that takes two operands, one on each side of the operator.
- A **Boolean expression** is one that evaluates as true or false.
- To **cast** a value is to transform it to another data type.
- A **check digit** is a digit added to a number (either at the end or the beginning) that validates the authenticity of the number.
- A **compound assignment operator** is an operator that performs two tasks, one of which is assignment.
- To **decrement** is to reduce by one.
- The **divide and assign operator** (/=) divides the operand on the left by the operand on the right.
- The **division operator** is the / symbol; it is a binary arithmetic operator used to perform division.
- An **explicit cast** is a deliberate cast; you can perform an explicit cast in one of two ways: by typing `static_cast<data type>` in front of an expression, or by using a type name within parentheses in front of an expression.
- The **hexadecimal numbering system** is a numbering system based on powers of 16.
- An **implicit cast** is the automatic cast or transformation that occurs when you assign a value of one type to a type with higher precedence.
- To **increment** is to increase by one.
- A **mixed expression** is one in which the operands have different data types.
- The **modulus and assign operator** (%=) finds the modulus when you divide the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
- The **modulus operator** is the % symbol; it is a binary arithmetic operator used to perform modulus. The modulus operator gives the remainder of integer division; it can be used only with integers.
- The **multiplication operator** is the * symbol; it is a binary arithmetic operator used to perform multiplication.
- The **multiply and assign operator** (*=) multiplies the left-hand operand by the right-hand operand.
- The **negative value operator** (–) is a unary operator that indicates a negative value.
- The **not operator** (**!**) reverses the true/false value of an expression.
- The **positive value operator** (+) is a unary operator that indicates a positive value.
- The **postfix increment operator** is ++ after a variable.
- The **prefix increment operator** is ++ before a variable.

- **Relational operators** are those that evaluate the relationship between operands.
- The **subtract and assign operator** (– =) subtracts the right-hand operand from the left-hand operand.
- The **subtraction operator** is the – symbol; it is a binary arithmetic operator used to perform subtraction.
- **Unary operators** are those that require only one operand.
- The **unifying type** is the data type of the value in an arithmetic expression to which all the types in the expression are converted.