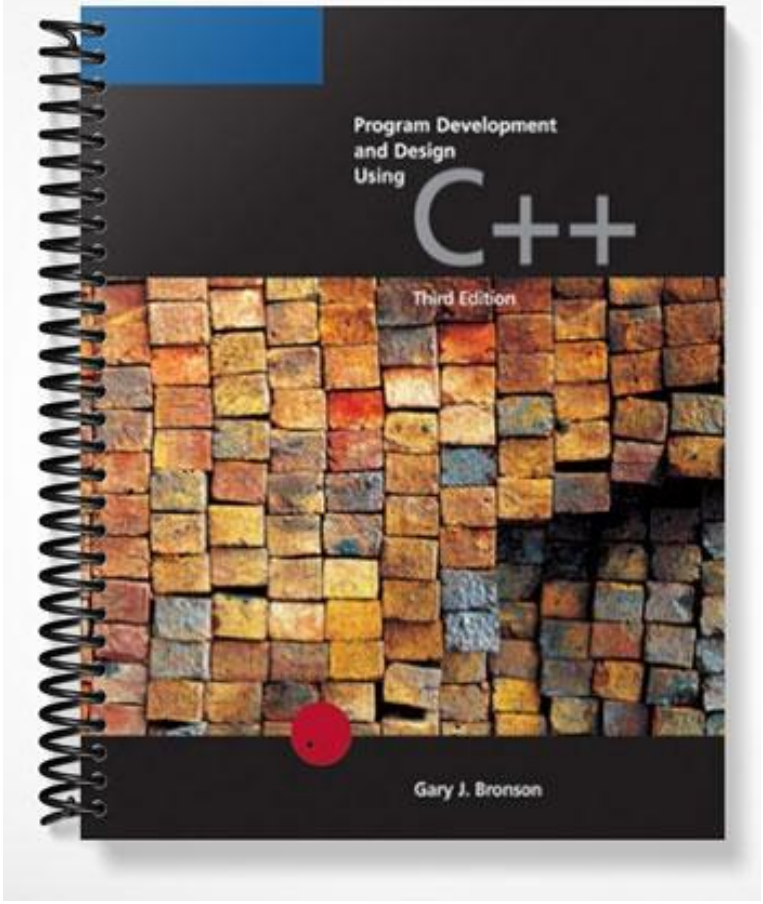
SOLUTIONS MANUAL



Chapter 2

Problem Solving Using C++

At a Glance

Instructor's Manual Table of Contents

- Overview
- Objectives
- Instructor Notes
- Quick Quizzes
- Discussion Questions
- Key Terms

Lecture Notes

Overview

This chapter begins with an introduction to C++, including discussions of the main() function and the cout object. Proper programming style is also discussed. The basic built-in data types are introduced and described, and an overview of arithmetic operators and operations is provided. An introduction to memory allocation and variable declaration is also given. Next, the software development procedure is reviewed, and applied to a handful of sample problems. Finally, the concept of abstract data types is introduced.

Chapter Objectives

Students should be able to describe:

- Introduction to C++
- Programming Style
- Data Types
- Arithmetic Operations
- Variables and Declaration Statements
- Applying the Software Development Procedure
- Problem Solving
- Introduction to Abstraction
- Common Programming Errors

Instructor Notes

Introduction to C++

A well-designed program depends on careful planning and execution if the final design is to accomplish its intended purpose.

A **Modular program's** structure consists of interrelated segments, arranged in a logical and easily understandable order to form an integrated and complete unit**.** Modular programs are easier to develop, correct, and modify than programs constructed in some other manner. The smaller segments used to construct a modular program are referred to as **modules.** Each module is designed and developed to perform a specific task and is really a small subprogram all by itself.

With modular construction, the program can be designed before any module is written.

In C++, modules can be either classes or functions. The interface of the function to the outside world is its inputs and results. The process of converting the inputs to results is both encapsulated and hidden within the function.

A **class** contains both data and specific functions appropriate for manipulating the data. A class encapsulates both data and one or more sets of operations.

Good function (and class) design includes giving it a name that conveys its purpose. The names permissible for functions and classes are also used to name other elements of the C++ language and are collectively referred to as identifiers. **Identifiers** can be made up of any combination of letters, digits, or underscores (_) selected according to the following rules:

1. The first character of the name must be a letter or underscore (_).

2. Only letters, digits, or underscores may follow the initial letter. Blank spaces are not allowed; separate words in a name consisting of multiple words by capitalizing the first letter of one or more of the words.

3. An identifier cannot be one of the keywords listed in Table 2.1 in the text. (A keyword is a word that is set aside by the language for a special purpose.)

4. The maximum number of characters in an identifier is limited to 255 characters.

Teaching Tip	Examples of valid and invalid identifiers are given in the text.

A good function name should be a mnemonic. A **mnemonic** is a word or name designed as a memory aid.

The main() Function

To provide for the orderly placement and execution of modules, each C++ program must have one and only one function named *main()*. The *main()* function is referred to as a **driver function** because it drives or tells the other modules the sequence in which they are to execute.

Figure 2.4 in the text illustrates a structure for the *main()* function. The first line of the function is referred to as a **function header line.** A function header line contains three pieces of information:

1. What type of data, if any, is returned from the function
2. The name of the function
3. What type of data, if any, is sent into the function

The keyword before the function name defines the type of value the function returns when it has completed operating. Empty parentheses following the function name signify that no data are transmitted into the function when it is run. (Data transmitted into a function at run time are referred to as **arguments** of the function.) Braces, { and }, determine the beginning, and end, respectively, of the function body and enclose the statements making up the function. The statements inside the braces determine what the function does. Each statement inside the function must end with a semicolon (;).

Teaching Tip	Figure 2.4 in the text illustrates the pieces of the *main()* function, as well as the standard layout.

The cout Object

One of the most versatile and commonly used objects provided in C++ is named *cout*. The *cout* object displays on the monitor whatever is passed to it.

Data are passed to the *cout* object by enclosing the text within quotation marks and putting the insertion (“put to”) symbol, <<, before the message and after the object’s name.

Initial comment lines at the beginning of a program, at a minimum, should provide the file name under which the source code is saved, a short program description, the name of the programmer, and the date that the program was last modified.

Preprocessor commands begin with a pound sign (#) and perform some action before the compiler translates the source program into machine code. The *#include* preprocessor command causes the contents of the named file to be inserted wherever the *#include* command appears in the program.

The classes *istream* and *ostream* provide the data declarations and methods used for data input and output, respectively. The *iostream* file is referred to as a **header file** because a reference to it is always placed at the top, or head, of a C++ program using the *#include* command.

The statement

```
using namespace std;
```

tells the compiler where to look to find the header files in the absence of any further explicit designation. Using namespaces effectively permits you to create your own classes and functions with the same names as those provided by the standard library, and place them in differently named namespaces.

cout is an object of a prewritten class; it is available for use just by activating it.

When a string of characters is passed to *cout*, the object sees to it that the string is correctly displayed on the monitor. Strings in C++ are any combination of letters, numbers, and special characters enclosed in double quotes "string in here".

The two characters \ and n, when used together, are called a *newline escape sequence.* They tell *cout* to send instructions to the display device to move to a new line. In C++, the backslash (\) character provides an “escape” from the normal interpretation of the character following it by altering the meaning of the next character.

Teaching Tip	C++ also supports C’s output functions, such as *printf()*. These can provide for better control of the format of output, but are generally more difficult to use.

Programming Style

C++ programs start execution at the beginning of the *main()* function. Although the *main()* function must be present in every C++ program, C++ does not require that the word main, the parentheses ( ), or the braces { } be placed in any particular form.

The following shows the standard form that should be used for writing the main function:

```
int main()
{
   program statements in here;

   return 0;
}
```

More than one statement can be put on a line, or one statement can be written across lines. Except for strings, double quotes, identifiers, and keywords, C++ ignores all white space.

Indentation is a sign of good programming.

Comments

Comments are explanatory remarks made within a program. Comments can be very helpful in clarifying what the complete program is about, what a specific group of statements is meant to accomplish, or what one line is intended to do.

A line comment begins with two slashes (//) and continues to the end of the line.

Comments that span two or more lines are, however, more conveniently written as block comments rather than as multiple-line comments. Such comments begin with the symbols /* and end with the symbols */.

In C++, a program's structure is intended to make the program readable and understandable, making use of extensive comments unnecessary. This is reinforced if function, class, and variable names, described in the next chapter, are carefully selected to convey their meaning to anyone reading the program.

Obscure code with no comments is a sure sign of bad programming when the program must be maintained or read by others. Excessive comments are also a sign of bad programming because they imply that insufficient thought was given to making the code self-explanatory.

Typically, any program that you write should begin with a set of initial program comments that include a short program description, your name, and the date that the program was last modified.

Data Types

The objective of all programs is ultimately to process data. Central to this objective is the classification of data into specific types. C++ allows only certain operations to be performed on certain data types.

A **data type** is defined as a set of values *and* a set of operations that can be applied to these values.

C++ categorizes data types into one of two fundamental groupings: built-in data types and class data types. A **class data type** is a programmer-created data type.

A **built-in data type** is one that is provided as an integral part of the C++ compiler and requires no external C++ code. Built-in data types, which are also referred to as **primitive** types, consist of the basic numerical types and basic mathematical operations.

A **literal** is an acceptable value for a data type. Another name for a literal is a **literal value**, or **constant**.

Integer Data Types

C++ provides nine built-in integer data types. These are shown in Figure 2.8 in the text. The essential difference among the various integer data types is the amount of storage used for each type, which directly affects the range of values that each type is capable of representing. The three most important types that are used almost exclusively in the majority of applications are the int, char, and bool data types.

The int Data Type

The set of values supported by the int data type are whole numbers (integers). An integer value consists of digits only and can optionally be preceded by either a plus (+) or minus (-) sign. Different compilers have their own internal limit on the largest and smallest integer values that can be stored in each data type.

The char Data Type

The *char* data type is used to store individual characters. Characters include the letters of the alphabet (both uppercase and lowercase), the ten digits 0 through 9, and special symbols such as + $. , _!. A single character value is any one letter, digit, or special symbol enclosed by single quotes.

Character values are typically stored in a computer using either the ASCII or Unicode codes.

Teaching Tip	An **atomic data value** is a value that is considered a complete entity by itself and cannot be decomposed into a smaller data type.

The Escape Character

The backslash, \, has a special meaning in C++. It is referred to as the **escape character**. When this character is placed directly in front of a select group of characters, it tells the compiler to escape from the way these characters would normally be interpreted. The combination of a backslash and these specific characters is called an **escape sequence**.

C++'s most common escape sequences are listed in Table 2.4 in the text.

The bool Data Type

The *bool* data type is used to represent Boolean (logical) data. As such, this data type is restricted to one of two values: true or false. This data type is most useful when a program must examine a specific condition and, as a result of the condition being either true or false, take a prescribed course of action.

Determining Storage Size

C++ offers an operator named *sizeof()* that provides the number of bytes used to store values for any data type name included within the operator's parentheses.

Each time the compiler encounters the newline escape sequence, either as a single character or as part of a string, it is translated as a single character that forces the display to start on a new line.

Teaching Tip	Both *'\n'* and *"\n"* are recognized by the compiler as containing the newline character. The difference is in the data types being used. Formally, *'\n'* is a character literal, while *"\n"* is a string literal.

Signed and Unsigned Data Types

A **signed data type** is one that permits storing negative values in addition to zero and positive values. The int data type is a signed data type.

An **unsigned data type** is one that provides only for non-negative values. Both the char and bool data types are unsigned data types.

Teaching Tip	Except for the Boolean type, all of C++'s built-in data types are direct carryovers from the C procedural language. Therefore programs using only individual built-in types will not be object-oriented programs.

Floating-Point Types

A **floating-point number**, which is also called a **real number**, can be the number zero or any positive or negative number that contains a decimal point.

C++ supports three floating-point data types: *float*, *double*, and *long double*. The difference between these data types is the amount of storage that a compiler uses for each type. Most compilers use twice the amount of storage for *double*s as for *float*s, which allows a *double* to have approximately twice the precision of a *float*. For this reason, a *float* value is sometimes referred to as a **single precision** number and a *double* value as a **double-precision** number.

A *float* literal is indicated by appending either an f or F after the number and a *long double* literal is created by appending either an l or L to the number.

Teaching Tip	Floating-point numbers default to *double*s. A statement such as: float x = 15.0; may give a compiler warning because you are assigning a *double* to a *float*, which is a narrower type.

Exponential Notation

Floating-point numbers can also be written in exponential notation, which is commonly used to express both very large and very small values in compact form.

In exponential notation, the letter e stands for exponent. The number following the e represents a power of 10 and indicates the number of places the decimal point should be moved to obtain the standard decimal value. The decimal point is moved to the right if the number after the e is positive or moved to the left if the number after the e is negative.

Teaching Tip	Information on the computer representation of real numbers can be found at: www.eskimo.com/~scs/cclass/progintro/sx5.html

Teaching Tip	In computer programming, precision can refer either to the accuracy of a number or the amount of significant digits in the number, where significant digits are defined as the number of clearly correct digits plus 1.

Quick Quiz

1. True or False: C++ compilers do not distinguish between uppercase and lowercase letters.
   Answer: False

2. True or False: A C++ program must have exactly one *main()* function.
   Answer: True

3. ____ are explanatory remarks made within a program.
   Answer: Comments

4. The ____ data type is restricted to one of two values: true or false
   Answer: *bool*

Arithmetic Operations

Integers and real numbers can be added, subtracted, multiplied, and divided. You can also add character data to, or subtract it from, both character and integer data to produce useful results, because characters are stored using integer storage codes.

The operators used for arithmetic operations are called **arithmetic operators**, and are as follows:

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus division	%

Binary operators require two operands to produce a result. An **operand** can be either a literal value or an identifier with an associated value. A **simple binary arithmetic expression** consists of a binary arithmetic operator connecting two literal values in the form:

literalValue *operator* literalValue

You can use *cout* to display the value of any arithmetic expression on the console screen.

In addition to displaying a numerical value, *cout* can display a string identifying the output. For instance:

```
cout << "The sum of 6 and 15 is " << (6 + 15);
```

displays

```
The sum of 6 and 15 is 21
```

When multiple insertions are made to *cout* the code can be spread across multiple lines. Only one semicolon, however, must be used, which is placed after the last insertion and terminates the complete statement.

When you allow such a statement to span multiple lines, a few rules must be followed:

1. A string contained within double quotes cannot be split across lines

2. The terminating semicolon appears only on the last line. Multiple insertion symbols can always be placed within a line.

A **manipulator** is an item used to manipulate how the output stream of characters is displayed. The *endl* manipulator first causes a newline character (‘\n’) to be inserted into the display and then forces all of the current insertions to be displayed immediately.

Teaching Tip	A fairly complete list of C++ operators can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclang/html/_langref_c.2f.c.2b2b_.operators.asp

Teaching Tip	You should use the *\n* escape sequence whenever it can be included within an existing string, and use the *endl* manipulator whenever a *\n* would appear by itself or to formally signify the end of a specific group of output display.

Expression Types

An **expression** is any combination of operators and operands that can be evaluated to yield a value. An expression that contains only integer values as operands is called an **integer expression**, and the result of the expression is an integer value. An expression containing only floating-point values as operands is called a **floating-point expression**, and its result is a floating-point value. An expression containing both integer and floating-point values is called a **mixed mode expression**. In a mixed mode expression:

1. If both operands are integers, the result of the operation is an integer.

2. If one operand is a real value, the result of the operation is a double-precision value.

Integer Division

The division of two integer values can produce rather strange results for the unwary. For example, the expression 15/2 yields the integer result 7. Because integers cannot contain a fractional part, a value of 7.5 cannot be obtained. The fractional part obtained when two integers are divided, that is, the remainder, is always truncated.

The **modulus operator (**or **remainder operator)**, *%*, captures the remainder when an integer number is divided by an integer.

Negation

A **unary operator** operates on a single operand. One of these unary operators uses the same symbol as binary subtraction (-). The minus sign in front of a single numerical value negates (reverses the sign of) the number.

Operator Precedence and Associativity

C++ requires you to follow certain rules when writing expressions containing more than one arithmetic operator.

1. Two binary arithmetic operator symbols must never be placed side by side.

2. Parentheses may be used to form groupings, and all expressions enclosed within parentheses are evaluated first. This permits parentheses to alter the evaluation to any desired order.

3. Sets of parentheses may also be enclosed by other parentheses. The number of closing parentheses [ )] must always equal the number of opening parentheses [( ] so that there are no unpaired sets.

4. Parentheses cannot be used to indicate multiplication

Expressions within parentheses are always evaluated first. Expressions containing multiple operators, both within and without parentheses, are evaluated by the **precedence** of the operators. There are three levels of precedence:

- P1—All negations are done first.

- P2—Multiplication, division, and modulus operations are computed next. Expressions containing more than one multiplication, division, or modulus operator are evaluated from left to right as each operator is encountered.

- P3—Addition and subtraction are computed last. Expressions containing more than one addition or subtraction are evaluated from left to right as each operator is encountered.

Teaching Tip	A fairly complete operator precedence table can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclang/html/_pluslang_c.2b2b_.operators.asp

Variables and Declaration Statements

All integer, floating-point, and other values used in a computer program are stored and retrieved from the computer's memory unit.

Symbolic names called **variables** are used in place of actual memory addresses. A variable is simply a name chosen by the programmer that is used to refer to computer storage locations. The term **variable** is used because the value stored in the variable can change, or vary.

In C++, the selection of variable names is left to the programmer, as long as the following rules are observed:

1. The variable name must begin with a letter or underscore (_) and may contain only letters, underscores, or digits. It cannot contain any blanks, commas, or special symbols, such as ( ) & , $ # . ! \ ?. Use capital letters to separate names consisting of multiple words.

2. A variable name cannot be a keyword.

3. The variable name cannot consist of more than 255 characters.

Variable names should be mnemonics that give some indication of the variable's use.

Assignment statements tell the computer to assign a value into a variable. Assignment statements always have an equal (=) sign and one variable name immediately to the left of this sign. The value on the right of the equal sign is determined first, and this value is assigned to the variable on the left of the equal sign.

Teaching Tip	Students should be introduced to pictorial representations of memory and storage at this point (if they have not already been introduced).

Declaration Statements

Naming a variable and specifying the data type that can be stored in it is accomplished using a **declaration statement.** A declaration statement has the general form:

dataType variableName;

where dataType designates a valid C++ data type and *variableName* is a user-selected variable name.

Most declarations are typically grouped together and placed immediately after the function's opening brace. A variable must be declared before it can be used.

When a variable name is sent to cout, the value stored in the variable is placed on the output stream and displayed.

Multiple Declarations

Variables having the same data type can always be grouped together and declared using a single declaration statement. The common form of such a declaration is:

dataType variable List;

Declaration statements can also be used to store an initial value into declared variables (**initialize** the variables**)**.

Inserting a blank line after the variable declarations placed at the top of a function body is a good programming practice. It improves both a program's appearance and its readability.

Variable declarations may be freely intermixed and even contained within other statements; the only requirement is that a variable must be declared prior to its use.

Memory Allocation

The computer can allocate storage for a variable only after knowing the variable's data type. Variable declarations provide this information. They can be used to force the compiler to reserve sufficient physical memory storage for each variable. Declaration statements used for this hardware purpose are also called **definition statements**.

Displaying a Variable's Address

Every variable has three major items associated with it: its data type, the actual value stored in the variable, and the address of the variable. The value stored in the variable is referred to as the variable's contents, while the address of the first memory location used for the variable constitutes its address. The number of locations used for a variable depends on its data type.

Programmers are usually concerned only with the value assigned to a variable and give little attention to where the value is stored (its address).

To determine the address of a variable named *num*, we can use C++'s address operator, &, which means "the address of." Except when used in a declaration statement, the address operator placed in front of a variable's name refers to the address of the variable. For example, *&num* means the address of *num*, *&total* means the address of *total*, and *&price* means the address of *price*.

Applying the Software Development Procedure

In this section, we show how the steps presented in Section 1.3 of the text are applied in practice when converting programming problems into working C++ programs.

Step 1: Analyze the Problem

The analysis can consist of up to two parts. *Basic analysis* consists of extracting the complete input and output information supplied by the problem.

1. Determine and understand the desired output items that the program must produce

2. Determine the required input items

Together, these two items are referred to as the problem's input/output (I/O). The I/O must be determined before selecting an initial algorithm for transforming the inputs into the desired outputs. At this point, it is sometimes necessary and/or useful to perform a hand calculation to verify that the output can indeed be obtained from the inputs.

An *extended analysis* involves obtaining additional information about the problem.

Step 2: Develop a Solution

In this step, you must settle on an algorithm for transforming the input items into the desired outputs and refine it as necessary so that it adequately defines all of the features that you want.

When designing a solution, the approach we take is often referred to as the **top-down approach**, which starts with the most general solution and refines it in a manner such that the final program solution consists of clearly defined tasks that can be accomplished by individual program functions.

Step 3: Code the Solution

At this point, you write the C++ program that corresponds to the solution developed in Step 2.

Step 4: Test and Correct the Program

This is done by means of selected test data and is used to make corrections to the program when errors are found. One set of test data that should always be used is the data used in your previous hand calculation.

In the text, the four steps are applied to the following problem:

> The circumference, *C,* of a circle is given by the formula $C = 2\pi r$, where π is the constant 3.1416 (accurate to four decimal places) and *r* is the radius of the circle. Using this information, write a *C++* program to calculate the circumference of a circle that has a 2-inch radius.

An **input item** is the name of an input quantity, whereas an **input value** is a specific number or quantity for the input item.

Focus on Problem Solving

In this section, the software development procedure presented in the previous section is applied to two specific programming problems.

Problem 1: Pendulum Clocks

The relationship between the time to complete one swing and the length of the pendulum is given by the formula

$$\text{length} = g[\text{time} / (2\pi)]^2$$

Using the given formula, we will write a C++ program that calculates and displays the length of a pendulum needed to produce a swing that will be completed in 1 second.

The output is the length of the pendulum. The problem specifies that the value be displayed in units of inches. The input items required to solve for the length are the time to complete one swing, the gravitational constant, *g,* and π.

The solution code is:

```
#include <iostream>
using namespace std;

int main()
{
    double time, length, pi;

    pi = 3.1416;
    g = 32.2;
    time = 1.0;
    length = 12.0 * g * time/(2.0*pi) * time/(2.0*pi);
    cout << "The length is " << length << " inches\n";

    return 0;
}
```

Problem 2: Telephone Switching Networks

The number of direct lines needed to maintain a directly connected network for *n* telephones is given by the formula:

lines = n(n - 1)/2

Using the given formula, we will write a C++ program that determines the number of direct lines required for 100 telephones and the additional lines required if 10 new telephones were added to the network.

Two outputs are required: the number of direct lines for 100 telephones and the additional number of lines needed when 10 new telephones are added to the existing network. The input item required for this problem is the number of telephones, which is denoted as n in the formula.

The solution code is:

```
#include <iostream>
using namespace std;

int main()
{
   int numin1, numin2, lines1, lines2;

   numin1 = 100;
   numin2 = 110;
   lines1 = numin1 * (numin1 - 1)/2;
   lines2 = numin2 * (numin2 - 1)/2;
   cout << "The number of initial lines is " << lines1 << ".\n";
   cout << "There are " << lines2 - lines1
        << " additional lines needed.\n";

   return 0;
}
```

Planning for Objects: An Introduction to Abstraction

An **abstraction** is an idea or term that identifies the general characteristics of a group of objects, independent of any one specific object in the group.

In programming terminology, a **data type** consists of *both* an acceptable range of values for a particular type and a set of operations that can be applied to those values. Thus, the integer data type not only defines a range of acceptable integer values but also defines what operations can be applied to those values.

A data type is an abstraction that consists of a set of values of a particular type and a set of operations that can be applied to those values. The set of allowed values is more formally referred to as the data type's **domain.**

Built-In and Abstract Data Types (ADTs)

All of the data types listed in Table 2.9 of the text are provided as part of the C++ language. They are formally referred to as **built-in** or **primitive** data types. In contrast to built-in data types, some programming languages permit programmers to create their own data types; that is, define a type of value with an associated domain and operations that can be performed on the acceptable values. Such user-defined data types are formally referred to as **abstract data types.** In C++, abstract data types are called **classes.**

Common Programming Errors

1. Omitting the parentheses after main.
2. Omitting or incorrectly typing the opening brace, {, that signifies the start of a function body.
3. Omitting or incorrectly typing the closing brace, }, that signifies the end of a function.
4. Misspelling the name of an object or function; for example, typing cot instead of cout.
5. Forgetting to close a string sent to cout with a double quote symbol.
6. Forgetting to separate individual data streams passed to cout with an insertion ("put to") symbol, <<.
7. Omitting the semicolon at the end of each C++ statement.
8. Adding a semicolon at the end of the #include preprocessor command.
9. Forgetting the \n to indicate a new line.
10. Incorrectly typing the letter O for the number zero (0), or vice versa. Incorrectly typing the letter l for the number 1, or vice versa.
11. Forgetting to declare all the variables used in a program.
12. Storing an inappropriate data type in a declared variable.
13. Using a variable in an expression before a value has been assigned to the variable.
14. Dividing integer values incorrectly.
15. Mixing data types in the same expression without clearly understanding the effect produced.

Quick Quiz

1. A(n) ____ is any combination of operators and operands that can be evaluated to yield a value.
   Answer: expression

2. The ____ operator captures the remainder when an integer number is divided by an integer.
   Answer: Modulus, remainder, or %

3. True or False: The + operator has a higher precedence than the * operators.
   Answer: False

4. ____ are symbolic names used in place of actual memory addresses.
   Answer: Variables

Discussion Questions

- Discuss how improper programming style might adversely affect a software development company (i.e., in terms of money and time).

- Discuss the ways in which the choice of data type in a program might affect how the program operates.

Key Terms

- **Abstraction: An** Idea or term that identifies the general qualities or characteristics of a group of objects, independent of any one specific object in the group
- **Argument:** Data transmitted into a function at run time
- **ASCII:** A standard code system that provides 256 codes for an English language-based character set plus codes for printer and display control, such as new line and printer paper-eject codes
- **Arithmetic operators:** Operators used for arithmetic operations
- **Assignment statement:** A statement that tells the computer to assign (store) a value into a variable
- **Associativity:** The order in which operators of the same precedence are evaluated
- **char:** A data type used to store individual characters
- **Character values:** A single character value is any one letter, digit, or special symbol enclosed by single quotes**,** typically stored in a computer using either the ASCII or Unicode codes
- **cout: An** output object that sends data given to it to the standard output display device
- **Data type:** A set of values *and* a set of operations that can be applied to these values
- **Declaration statement:** A statement that names a variable and specifies the data type that can be stored in it
- **Definition statement:** Statements that define or tell the compiler how much memory is needed for data storage

- **double: A** floating-point data type that typically uses 8 bytes of memory
- **Double-precision number:** Another name for the double floating-point data type
- **Escape sequence:** The combination of a backslash and certain characters used to alter the interpretation of certain characters
- **Expression:** Any combination of operators and operands that can be evaluated to yield a value
- **float:** A foating-point data type that typically uses 4 bytes of memory
- **Floating-point number:** A data type that can be the number zero or any positive or negative number that contains a decimal point
- **Function:** A single unit of code providing a special-purpose operation
- **Identifier: The** names permissible for functions, classes, and other elements of the C++ language
- **int:** A data type consisting of whole numbers, which are mathematically known as integers.
- **Integer value:** A data type that consists of digits only and can optionally be preceded by either a plus (+) or minus (-) sign
- **Keyword:** A word that is set aside by the language for a special purpose and can only be used in a specified manner
- **long:** A data type modifier (for int and double)
- **Manipulator:** An item used to manipulate how the output stream of characters is displayed
- **Mixed-mode expression:** An expression containing both integer and floating-point values
- **Mnemonic:** A word or name designed as a memory aid
- **Precedence:** The priority that indicates the order in which operators should be evaluated
- **Precision:** A concept that can refer either to the accuracy of a number or the amount of significant digits in the number
- **sizeof():** An operator used to determine the amount of storage reserved for variables
- **Variable:** A symbolic name used in place of an actual memory address