Chapter 2

Problem Solving Using C++

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

An integral part of a building's design is its structure, and the same is true for a program. Constructing well-designed C++ programs depends on careful planning and execution, if the final design is to ensure that the completed program accomplishes its intended purpose. A central element of this planning is using modular program design, which is explained in Section 2.1. In this chapter, your students also learn about different types of data and how to process them in the context of a complete C++ program.

Objectives

In this chapter, students will learn about:

- Modular programs
- Programming style
- Data types
- Arithmetic operations
- Variables and declaration statements
- Common programming errors

Teaching Tips

2.1 Introduction to C++

1. Introduce the concept of a modular program.

2. Introduce the concept of a function as a module.

3. Introduce the concept of a class in C++.

Teaching Tip	A commonly used definition of a class is a blueprint for an object that encapsulates both data and code.

4. Contrast a function and a class. The function has operations only, but the class also contains data.

5. Discuss the rules for naming identifiers.

Teaching Tip	Discuss the importance of using a good naming convention. Corporations often establish standards for naming and program structure that an employee must follow.

6. Discuss the purpose of keywords.

7. Introduce the concept of a mnemonic.

Teaching Tip	Stress the fact that C++ is a case-sensitive language, and describe how a simple error in capitalization is treated like a different variable.

The `main()` Function

1. Discuss the structure of a C++ program, using the **`main()`** function as the driver of the program.

2. Describe the use of the **`return`** and **`return 0`** statements.

3. Introduce the concept of arguments and how they are used.

Teaching Tip	Students often have difficulty understanding how a function operates. A good analogy is a food processor or blender. You put one or more ingredients into the top (the arguments), turn it on (process the statements in the body), and receive a finished product (the function's return value). Both the blender and the function use the inputs to produce the output.

The `cout` Object

1. Introduce the **`cout`** object for output displays.

2. Introduce the concept of **`#include`** preprocessor commands and header files.

3. Discuss the namespace concept as a means of qualifying an object.

Teaching Tip	Stress the importance of using prewritten classes instead of "reinventing the wheel."

4. Introduce the concept of a string and a delimiter.

Teaching Tip	Use the illustration of zip code as a string containing number characters. A useful design guideline is to determine whether a value with number characters will be used for computations; if not, it should be stored as a string. Note that using a string for zip code preserves a leading zero, whereas using a numeric data type would not.

5. Briefly introduce the concept of escape sequences.

<u>Quick Quiz 1</u>

1. Name the three parts of a function header.
   Answer: function's return data type, function's name, and data type of the arguments (if any)

2. What function acts as the driver for a C++ program?
   Answer: `main()` acts as the driver for a C++ program

3. What is the purpose of a namespace?
   Answer: The purpose is to define the section of source code to access when looking for prewritten classes or functions.

4. What is an escape sequence used for?
   Answer: An escape sequence provides special instruction codes to the compiler, such as formatting commands.

2.2 Programming Style

1. Discuss the importance of style in programming for readability and maintainability.

Teaching Tip	Standards given in this course are generally followed; however, students should consult their company's programming standards as well.

2. Although many compilers allow multiple statements per line, good standards dictate a single line for each instruction.

Teaching Tip	Remind students that they should always strive for good readability in their programs. Even if a compiler supports other styles, a good programmer always uses indentation appropriately, and keeps all declarations at the beginning of the function in which they are declared.

3. Point out that indentation is highly important for readability.

Teaching Tip	Indentation also helps the programmer by indicating where blocks of code start and end. This will become more evident as other types of C++ statements are introduced.

Comments

1. Introduce line comments and block comments.

Teaching Tip	Point out that comments are made by programmers for programmers. Comments in the code are never seen at runtime.

Quick Quiz 2

1. How does the C++ compiler handle white space?
   Answer: White space is ignored.

2. Why is it important to adhere to the standard form when writing code?
   Answer: It provides enhanced readability.

3. What is the purpose of a comment in a program?
   Answer: Comments provide information for any programmer who is working on the code.

2.3 Data Types

1. Introduce the concept of data types for storing different types of information.

2. Point out that while a programmer can always create his/her own class to represent data, the built-in (primitive) data types should be used when suitable.

3. Describe the two numerical data types, integer and floating point.

4. Introduce the concept of a literal value.

Integer Data Types

1. Briefly describe the nine integer data types.

Teaching Tip	Students often get apprehensive when faced with such a large choice of data types. Point out that only a subset of these is commonly used in most programs.

2. Discuss the uses of the **`int`** data type.

3. Introduce the **`char`** data type and the ASCII codes for characters.

Teaching Tip	Students are often confused about why character data is considered to be an integer. An explanation of the ASCII codes and their use with character data will resolve this confusion.

4. Describe the use of escape sequences, and introduce the most commonly used escape sequences.

5. Introduce the **`bool`** data type.

Determining Storage Size

1. Introduce the use of the **`sizeof`** operator.

Teaching Tip	Point out that there are implementation differences in data storage between different computer platforms and/or compilers. One should always check system documentation to understand how data is stored on the computer system that is to be used.

2. Introduce the use of the **`sizeof`** operator.

Signed and Unsigned Data Types

1. Introduce the concept of signed data types and unsigned data types.

2. Describe the uses of signed and unsigned data types. Point out that the data ranges differ when signs are or are not included in the data storage type.

3. Review the storage requirements of the different integer data types.

Floating-Point Types

1. Introduce floating-point data types and describe their use. Point out again the importance of the **`sizeof`** operator for determining storage use.

2. Discuss the difference between a single-precision and a double-precision number.

Quick Quiz 3

1. What are the two general types of numerical data?
   Answer: integer and floating-point

2. How do signed and unsigned data types differ?
   Answer: Signed data types permit storing negative values, whereas unsigned data types only hold non-negative values.

3. Which data type should be used for a variable that will only hold true or false values?
   a. **`int`**
   b. **`long int`**
   c. **`bool`**
   d. **`char`**
   Answer: c

2.4 Arithmetic Operations

1. Discuss the arithmetic operations supported by C++. Point out the use of the asterisk for multiplication (not an x).

Teaching Tip	Students may have forgotten the meanings of the terms *operator* and *operand*. It is a good idea to review these terms at this point.

Expression Types

1. Introduce the concept of an arithmetic expression.

2. Differentiate integer and floating-point expressions.

3. Discuss mixed-mode expressions, along with the associated rules that define the resulting data type.

Teaching Tip	Explain that the computer requires that all data be the same data type in order to do an arithmetic operation, so internal conversions may be occurring. These conversions may impact the final results. Stress the importance of good data type selections when programming to avoid nasty surprises.

Integer Division

1. Describe integer division, and when it might be appropriate to use it.

Teaching Tip	A good example for using integer division is to determine how many patterns can be cut from a sheet of material. A partial pattern is useless.

2. Describe the modulus operation, and when it might be appropriate to use it.

Teaching Tip	A common use of modulus is to determine if a number is odd or even.

Negation

1. Introduce the concept of negation (or flipping the sign). Point out that this is a unary command because only one operand is required.

Operator Precedence and Associativity

1. Describe the rules of using operators and parentheses within expressions.

2. Describe the order of precedence of operators.

3. Describe associativity as the means of determining what operation is performed first when several operators are at the same level of precedence.

Teaching Tip	Are you having trouble remembering the order of precedence? Just use parentheses to force specific parts of your expression to be evaluated in the desired order.

<u>Quick Quiz 4</u>

1. What is an expression?
   Answer: An expression is any combination of operators and operands that can be evaluated to yield a value.

2. What happens to fractional results in integer division?
   Answer: The fractional part is truncated.

3. How can precedence be controlled in expression evaluation?
   Answer: Precedence can be controlled by using parentheses to form groupings.

2.5 Variables and Declaration Statements

1. Review the concept of data storage and retrieval from memory locations.

2. Remind students that assignment statements store a value in a variable.

Declaration Statements

1. Remind students that variable names must follow the identifier-naming rules introduced earlier.

2. Introduce the concept of declaring a variable. Point out that declaring a variable sets up the memory location in which the variable's value will be stored.

Multiple Declarations

1. Point out that variables of the same data type may be declared in a single statement, although most programming standards prefer a separate statement for each declaration.

2. Review the format of multiple declarations with your students.

3. Highlight the concept of variable initialization, and that a variable is assigned a default value if none is explicitly given upon declaration.

Memory Allocation

1. Introduce the concept of a variable as a named memory location.

Teaching Tip	Stress the difference between assignment statements and algebraic statements. Note that an assignment statement cannot have an expression on the left side.

.

2. Introduce the concept of a definition statement as a declaration combined with an assignment statement. Explain the term "initialization of the variable."

Displaying a Variable's Address

1. Introduce assignment statements.

2. Remind students that all variables must be declared before they can be used.

3. Introduce the address operator **&**, and inform students that the ability to use the actual memory address of the variable will be explored in Chapters 6 and 12.

Quick Quiz 5

1. What is a variable?
   Answer: A variable is a symbolic name for a storage location.

2. What is the purpose of a declaration statement?
   Answer: A declaration statement is used to name a variable and specify the data type that can be stored there.

3. When is a variable said to be initialized?
   Answer: A variable is said to be initialized when a declaration statement is used to store a value in a variable.

2.6 A Case Study: Radar Speed Traps

1. Review the four steps of the software development procedure. Step 1: Point out that sometimes a basic analysis is sufficient, but for other requirements, an extended, in-depth analysis may be required.

Teaching Tip	In a real-world software development project, it is not unusual for the requirements analysis phase to take several months or longer, depending on the complexity of the requirements.

2. Step 2: Discuss briefly the top-down approach to designing a solution.

Teaching Tip	The top-down approach of layered refinement helps to keep you focused on the "big picture" of what you are trying to accomplish.

3. Step 3: This is where the actual coding occurs. Stress the importance of completing a thorough analysis and design *before* starting to write code.

4. Step 4: Testing is designed to ensure that the finished software product meets the requirements and functions correctly.

5. Walk students through the four-step software development procedure for each of the two sample applications.

Teaching Tip	Either of these two applications could serve as an extended class discussion topic.

Quick Quiz 6

1. What are the four steps in the software development procedure?
   Answer: analysis; development; coding; testing & correction.

2.7 Common Programming Errors

1. Discuss each of the listed programming errors.

Quick Quiz 7

1. True or False: A variable must be declared before it can be used.
   Answer: True

2. What character is used to signify the start of a function body, and is often omitted?
   Answer: the **`{`** character

Class Discussion Topics

1. Why are programming standards such as naming conventions, indenting, and use of comments important to an organization? Considering that following such conventions may actually require more lines of code, is there a "return on investment" for following these standards?

2. Discuss the types of problems that could occur from using the wrong data type for variables. What are some possible consequences if this occurs in software used to run robotic assembly lines, space missions, medical equipment, or research work?

Additional Projects

1. Have students create a small C++ program using the **`sizeof`** operator to display the storage size for each type of numeric data type. If possible, have them run the program on a Windows-based system and on a Unix-based system and compare the results from each platform.

2. Have students interview a working engineer, a scientist, or some other research professional who uses C++ in his/her work. Ask questions such as:
   - Why do you use C++?
   - What are the advantages and disadvantages of using C++?
   - How long did it take you to become proficient using C++?
   - What is your most common programming error?
   - What advice do you have for students just starting to learn C++?

Additional Resources

1. Beginner's C++ tutorial:
   http://www.intap.net/~drw/cpp/

2. Another useful C++ tutorial:
   http://www.cplusplus.com/doc/#tutorial

Key Terms

- **Arguments:** Data transmitted to a function at runtime
- **Arithmetic operators:** The operators used for arithmetic operations in the C++ context
- **Assignment statement:** A C++ language statement that tells the computer to store a value in a variable
- **Associativity:** The order in which operators with the same priorities are evaluated in a C++ expression
- **Atomic data type:** A data type that supports only atomic data values
- **Atomic data value:** A value that is considered a complete entity and which can't be decomposed into a smaller data type supported by the language
- **Binary operators:** Arithmetic operators that require two operands
- **Block comment:** Text in a C++ language source code file that represents a code comment, begins with the symbols /*, ends with the symbol, */, may extend over multiple lines, and is not translated into assembler by the compiler
- **Built-in data type:** A data type provided as part of the C++ compiler and that requires no external C++ code
- **Case-sensitive:** Upper- and lowercase characters are considered distinct characters
- **Class:** A small machine that contains both data and functions for manipulating that data, and containing elements for input, output, and processing
- **Class data type:** A programmer created (or abstract) data type
- **Comment:** An explanatory remark in the source code of a program that is visible only to the programmer
- **Constant:** A synonym for a literal
- **Data type:** A set of values and a set of operations that can be applied to these values
- **Declaration statement:** A C++ language statement in the general form `dataType variableName;` and which names a variable and specifies the variable's data type
- **Definition statement:** Statement that uses information provided by a variable declaration to define or tell the compiler how much memory is needed for data storage and to request memory be allocated for that purpose
- **Double-precision number:** A float value, which is a data type that holds a size-limited defined range of all real values in storage that is twice the width of a single-precision number
- **Driver function:** A function that defines the a set of functions to be executed, and the sequence in which to execute these functions
- **Escape sequence:** A combination of a backslash and one of a set of escape sequence characters, interpreted as a special character by the compiler
- **Expression:** Any combination of operators and operands that can be evaluated to yield a value

- **Floating-point expression:** An expression containing only floating point values
- **Floating-point number:** a synonym for the term “real number”
- **Function:** A small machine that transforms the data it receives into a finished product
- **Header file:** A prewritten programming file that can be included in a program you are writing
- **Identifier:** A name for an element of the programming language
- **Initialized:** Storing an initial value in a variable, often in a declaration statement
- **Integer:** Whole numbers; the number zero or any positive or negative number without a decimal point
- **Integer expression:** An expression containing only integer values
- **Keyword:** A reserved word in a programming language that has a special purpose to the compiler or interpreter
- **Line comment:** Text in a C++ language source code file that represents a code comment, begins with two slashes (`//`), continues to the end of the source file line, and is not translated into assembler by the compiler
- **Literal:** An acceptable value for a data type where the value identifies itself
- **Literal Value:** A synonym for a literal
- **Manipulator:** An item used to change how an output stream is displayed
- **Mixed-mode expression:** An expression containing integer and floating-point values
- **Mnemonic:** A word designed to be a memory aid; often composed of first initials of words in a phrase
- **Modular program:** Program with a structure consisting of interrelated segments (modules) arranged in a logical and easily understandable order to form an integrated and complete program unit
- **Module:** A segment of a computer program, usually designed to be reusable
- **Modulus operator:** This operator captures the remainder when an integer is divided by another integer
- **Namespace:** A section of source code the compiler accesses when it is looking for prewritten classes or functions
- **Newline escape sequence:** The characters \ and n, which, when used together, indicate that the output device should move to the beginning of a new line
- **Operand:** A literal value or an identifier with an associated value
- **Precedence:** The order in which operators with different priorities are evaluated in a C++ expression
- **Precision:** Numerical accuracy
- **Preprocessor commands:** Commands in source code that start with the pound sign (`#`) and perform an action before the compiler translates the source code into machine code
- **Primitive type:** A synonym for a built-in data type
- **Real number:** Any number zero, greater than zero, or less than zero, that contains a decimal point
- **Significant digits:** The number of clearly correct digits plus 1
- **Simple binary arithmetic operation:** A binary operator connecting two literal values in the form *`literalValue operator literalValue`*
- **Single-precision number:** A float value, which is a data type that holds a size-limited defined range of all real values in storage that is half the width of a double-precision number
- **String:** In C++, any combination of letters, numbers, or special characters enclosed in quotation marks
- **Unary operators:** Operators that perform their operation on a single operand

- **Unsigned data type:** A data type that provides for only non-negative values, meaning all values greater than or equal to zero
- **Variable:** An identifier that refers to a memory location
- **White space:** Any combination of blank spaces, tabs, or new lines