Chapter 2: Problem Solving Using C++
In this chapter, you will learn about:

- Modular programs
- Programming style
- Data types
- Arithmetic operations
- Variables and declaration statements
- Common programming errors
• **Modular program**: A program consisting of interrelated segments (or **modules**) arranged in a logical and understandable form
  – Easy to develop, correct, and modify

• **Modules in C++** can be classes or functions
• **Function**: Accepts an input, processes the input, and produces an output
  – A function’s processing is encapsulated and hidden within the function

*Figure 2.2* A multiplying function
• **Class**: Contains both data and functions used to manipulate the data

• **Identifier**: A name given to an element of the language, such as a class or function
  – Rules for forming identifier names:
    • First character must be a letter or underscore
    • Only letters, digits, or underscores may follow the initial letter (no blanks allowed)
    • Keywords cannot be used as identifiers
    • Maximum length of an identifier = 1024 characters
• **Keyword**: A reserved name that represents a built-in object or function of the language

<table>
<thead>
<tr>
<th>auto</th>
<th>delete</th>
<th>goto</th>
<th>public</th>
<th>this</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>do</td>
<td>if</td>
<td>register</td>
<td>template</td>
</tr>
<tr>
<td>case</td>
<td>double</td>
<td>inline</td>
<td>return</td>
<td>typedef</td>
</tr>
<tr>
<td>catch</td>
<td>else</td>
<td>int</td>
<td>short</td>
<td>union</td>
</tr>
<tr>
<td>char</td>
<td>enum</td>
<td>long</td>
<td>signed</td>
<td>unsigned</td>
</tr>
<tr>
<td>class</td>
<td>extern</td>
<td>new</td>
<td>sizeof</td>
<td>virtual</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>overload</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>private</td>
<td>struct</td>
<td>volatile</td>
</tr>
<tr>
<td>default</td>
<td>friend</td>
<td>protected</td>
<td>switch</td>
<td>while</td>
</tr>
</tbody>
</table>

**Table 2.1**: Keywords in C++
• Examples of valid C++ identifiers:

    degToRad  intersect  addNums
    slope     bessell    multTwo
    findMax   density

• Examples of invalid C++ identifiers:

    1AB3   (begins with a number)
    E*6    (contains a special character)
    while  (this is a keyword)
Function names
- Require a set of parentheses at the end
- Can use mixed upper and lower case
- Should be meaningful, or be a mnemonic

Examples of function names:
```
easy()  c3po()  r2d2()  theForce()
```

Note that C++ is a case-sensitive language!
The main() Function

- Overall structure of a C++ program contains one function named main(), called the driver function.
- All other functions are invoked from main().

Figure 2.3 The main() function directs all other functions.
• **Function header line:** First line of a function, which contains:
  – The type of data returned by the function (if any)
  – The name of the function
  – The type of data that must be passed into the function when it is invoked (if any)

• **Arguments:** The data passed into a function

• **Function body:** The statements inside a function
  – enclosed in braces
The **main()** Function (continued)

- Each statement inside the function must be terminated with a semicolon
- **return**: A keyword causing the appropriate value to be returned from the function
- The statement `return 0` in the `main()` function causes the program to end
The `main()` Function (continued)

Figure 2.4  The structure of a `main()` function
The `cout` Object

- **cout object**: An output object that sends data to a standard output display device

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

int main()
{
    cout << "Hello there world!";

    return 0;
}
```
The cout Object (continued)

• Preprocessor command: Starts with a #
  – Causes an action before the source code is compiled into machine code

• \#include <file name>: Causes the named file to be inserted into the source code

• C++ provides a standard library with many pre-written classes that can be included

• Header files: Files included at the head (top) of a C++ program
• **using namespace <namespace name>:** Indicates where header file is located
  - Namespaces qualify a name
    - A function name in your class can be the same as one used in a standard library class
• **String:** Any combination of letters, numbers, and special characters enclosed in double quotes
• **Delimiter:** A symbol that marks the beginning and ending of a string; not part of the string
The cout Object (continued)

Program 2.2

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

int main()
{
    cout << "Computers, computers everywhere";
    cout << "\n as far as I can C";

    return 0;
}
```
The *cout* Object (continued)

- **Escape sequence:** One or more characters preceded by a backslash, \n
**Program 2.3**

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

int main()
{
    cout << "Computers everywhere\n as far as\n I can see";

    return 0;
}
```
• Good style calls for one C++ statement per line
• Opening and closing braces \{ \} for the function body should each be on separate lines
• Statements in the function body should be indented
• **Comments:** Explanatory remarks in the source code added by the programmer

• **Line comment:** Begins with `//` and continues to the end of the line

  • **Example:** `// this program displays a message`

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

    int main ()
    {
        cout << "Hello there world!"; //displays text
        return 0;
    }
    ```
Comments (continued)

• **Block comments**: comments that span across two or more lines
  – Begin with /* and end with */
  – Example:
    ```
    /* This is a block comment that spans
     across three lines */
    ```
Data type: A set of values and the operations that can be applied to these values

Two fundamental C++ data groupings:
- Class data type (a class): Created by the programmer
- Built-in data type (primitive type): Part of the C++ compiler

Figure 2.5 Built-in data types
## Data Types (continued)

<table>
<thead>
<tr>
<th>Built-in Data Type</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>+, -, *, /, %, =, ==, !=, &lt;=, &gt;=, sizeof(), and bit operations (see Chapter 15, available online)</td>
</tr>
<tr>
<td>Floating point</td>
<td>+, -, *, /, =, ==, !=, &lt;=, &gt;=, sizeof()</td>
</tr>
</tbody>
</table>

**Table 2.2**  Built-In Data Type Operations
• **Literal (constant):** An actual value
  – Examples:
    
    
    ```
    3.6 //numeric literal
    “Hello” //string literal
    ```

• **Integer:** A whole number

• **C++ has nine built-in integer data types**
  – Each provides different amounts of storage (compiler dependent)
Figure 2.6  C++ integer data types
• **int** data type: Whole numbers (integers), optionally with plus (+) or minus (−) sign
  – Example: 2, −5

• **char** data type: Individual character; any letter, digit, or special character enclosed in single quotes
  – Example: ‘A’
  – Character values are usually stored in ASCII code
### Table 2.3 The ASCII Uppercase Letter Codes

<table>
<thead>
<tr>
<th>Letter</th>
<th>ASCII Code</th>
<th>Letter</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>01000001</td>
<td>N</td>
<td>01001111</td>
</tr>
<tr>
<td>B</td>
<td>01000010</td>
<td>O</td>
<td>01001110</td>
</tr>
<tr>
<td>C</td>
<td>01000011</td>
<td>P</td>
<td>01010000</td>
</tr>
<tr>
<td>D</td>
<td>01000100</td>
<td>Q</td>
<td>01010001</td>
</tr>
<tr>
<td>E</td>
<td>01000101</td>
<td>R</td>
<td>01010010</td>
</tr>
<tr>
<td>F</td>
<td>01000110</td>
<td>S</td>
<td>01010011</td>
</tr>
<tr>
<td>G</td>
<td>01000111</td>
<td>T</td>
<td>01010100</td>
</tr>
<tr>
<td>H</td>
<td>01001000</td>
<td>U</td>
<td>01010101</td>
</tr>
<tr>
<td>I</td>
<td>01001001</td>
<td>V</td>
<td>01010110</td>
</tr>
<tr>
<td>J</td>
<td>01001010</td>
<td>W</td>
<td>01010111</td>
</tr>
<tr>
<td>K</td>
<td>01001011</td>
<td>X</td>
<td>01011000</td>
</tr>
<tr>
<td>L</td>
<td>01001100</td>
<td>Y</td>
<td>01011001</td>
</tr>
<tr>
<td>M</td>
<td>01001101</td>
<td>Z</td>
<td>01011010</td>
</tr>
</tbody>
</table>
• When storing the ASCII codes shown in Table 2.3 to represent text, each letter takes one byte of memory and is represented by the associated number from the chart.

**Figure 2.7** The letters BARTER stored inside a computer
• **Escape character:** The backslash, \\n  – Indicates an escape sequence

• **Escape sequence:** Tells compiler to treat the following characters as special instruction codes
• `bool` data type: Represents Boolean (logical) data
  – Restricted to two values: true or false
  – Useful when a program must examine a condition and take a prescribed course of action, based on whether the condition is true or false
A unique feature of C++ is that you can see where and how values are stored

- `sizeof()` operator provides the number of bytes used to store values of the data type named in the parenthesis
- Values returned by `sizeof()` are compiler dependent
Determining Storage Size (continued)

Program 2.5

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

int main()
{
    cout << "\nData Type" << " Bytes"
        << "\n---------" << "-----"
        << "\ni int" << " sizeof(int)"
        << "\nchar" << " sizeof(char)"
        << "\nbool" << " sizeof(bool)"
        << '\n';
    return 0;
}
```
Signed and Unsigned Data Types

- **Signed data type**: One that permits negative, positive, and zero values
- **Unsigned data type**: Permits only positive and zero values
  - An unsigned data type provides essentially double the range of its signed counterpart
## Signed and Unsigned Data Types (continued)

<table>
<thead>
<tr>
<th>Name of Data Type</th>
<th>Storage Size</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
<td>256 characters</td>
</tr>
<tr>
<td>bool</td>
<td>1</td>
<td>true (considered as any positive value) and false (which is a 0)</td>
</tr>
<tr>
<td>short int</td>
<td>2</td>
<td>-32,768 to +32,767</td>
</tr>
<tr>
<td>unsigned short int</td>
<td>2</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td>long int</td>
<td>4</td>
<td>-2,147,483,648 to +2,147,483,647</td>
</tr>
<tr>
<td>unsigned long int</td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>

Table 2.5 Integer Data Type Storage
Floating-Point Types

• **Floating-point number** (real number): Zero or any positive or negative number containing a decimal point
  – Examples:  +10.625  5.  -6.2
  – No special characters are allowed
  – Three floating-point data types in C++:
    • `float` (single precision)
    • `double` (double precision)
    • `long double`
## Floating-Point Types (continued)

### Table 2.6 Floating-Point Data Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Absolute Range of Values (+ and -)</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>1.40129846432481707 × 10⁻⁴⁵ to 3.40282346638528860 × 10³⁸</td>
</tr>
<tr>
<td>double and long double</td>
<td>8 bytes</td>
<td>4.94065645841246544 × 10⁻³²⁴ to 1.79769313486231570 × 10³⁰⁸</td>
</tr>
</tbody>
</table>
Floating-Point Types (continued)

- **float literal**: Append an `f` or `F` to the number
- **long double literal**: Append an `l` or `L` to the number
  - Examples:
    
    ```
    9.234       // a double literal
    9.234F      // a float literal
    9.234L      // a long double literal
    ```
C++ supports addition, subtraction, multiplication, division, and modulus division

Different data types can be used in the same arithmetic expression

Arithmetic operators are binary operators
  - Binary operators: Require two operands
  - Unary operator: Requires only one operand
  - Negation operator (−): Reverses the sign of the number
## Arithmetic Operations (continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
</tr>
<tr>
<td>Modulus division</td>
<td>%</td>
</tr>
</tbody>
</table>
#include <iostream>
using namespace std;

int main()
{
    cout << "15.0 plus 2.0 equals " << (15.0 + 2.0) << endl
    << "15.0 minus 2.0 equals " << (15.0 - 2.0) << endl
    << "15.0 times 2.0 equals " << (15.0 * 2.0) << endl
    << "15.0 divided by 2.0 equals " << (15.0 / 2.0) << endl;

    return 0;
}
Expression Types

- **Expression**: Any combination of operators and operands that can be evaluated to yield a value.
- If all operands are the same data type, the expression is named by the data type used (integer expression, floating-point expression, etc.).
- **Mixed-mode expression**: Contains integer and floating-point operands.
  - Yields a double-precision value.
Integer Division

• Integer division: Yields an integer result
  – Any fractional remainders are dropped (truncated)
  – Example: \( 15/2 \) yields 7

• Modulus (remainder) operator: Returns only the remainder
  – Example: \( 9 \% 4 \) yields 1
Operator Precedence and Associativity

• Rules for writing arithmetic expressions:
  – Never place two consecutive binary arithmetic operators side by side
  – Use parentheses to form groupings
    • Contents within parentheses are evaluated first
  – May nest parentheses within other parentheses
    • Evaluated from innermost to outermost
  – Use the * operator for multiplication, not parentheses
Expressions with multiple operators are evaluated by precedence of operators:

- All negations occur first
- Multiplication, division, and modulus are next, from left to right
- Addition and subtraction are last, from left to right
• **Associativity**: the order in which operators of the same precedence are evaluated

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary –</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ –</td>
<td>Left to right</td>
</tr>
</tbody>
</table>

**Table 2.8** Operator Precedence and Associativity
Variables and Declaration Statements

• **Variable:** All integer, float-point, and other values used in a program are stored and retrieved from the computer's memory.

• Each memory location has a unique address.

**Figure 2.8** Enough storage for two integers.
Variables and Declaration Statements (continued)

• **Variable**: Symbolic identifier for a memory address where data can be held

• Use identifier naming rules for variable names

![Diagram](image)

**Figure 2.9** Naming storage locations
• **Assignment statement**: Used to store a value into a variable

• Value of the expression on the right is assigned to the memory location of the variable on the left side

  – Examples:

    num1 = 45;
    num2 = 12;
    total = num1 + num2;

Variables and Declaration Statements (continued)
• **Declaration statement**: Specifies the data type and identifier of a variable; sets up the memory location
  – **Syntax**: `dataType variableName;`

• Data type is any valid C++ data type
  – **Example**: `int sum;`

• Declarations may be used anywhere in a function
  – Usually grouped at the opening brace
Variables and Declaration Statements (continued)

• **Character variables:** Declared using the `char` keyword

• **Multiple variables of the same data type can be declared in a single declaration statement**
  – Example:
    
    ```
    double grade1, grade2, total, average;
    ```

• **Variables can be initialized in a declaration**
  – Example:
    
    ```
    double grade1 = 87.0
    ```

• **A variable must be declared before it is used**
Variables and Declaration Statements (continued)

Program 2.7a

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

int main()
{
    double grade1 = 85.5;
    double grade2 = 97.0;
    double total, average;

    total = grade1 + grade2;
    average = total/2.0; // divide the total by 2.0
    cout << "The average grade is " << average << endl;

    return 0;
}
```
Memory Allocation

- **Definition statement**: A declaration that defines how much memory is needed for data storage
- **Three items associated with each variable**:
  - Data type
  - Actual value stored in the variable (its contents)
  - Memory address of the variable
- **Address operator** (`&`) provides the variable’s address
Memory Allocation (continued)

• Declaring a variable causes memory to be allocated based on the data type

float slope;

“Tag” the first byte of reserved storage with the name slope

Reserve enough room for a single-precision number

Figure 2.10b Defining the floating-point variable named slope
#include <iostream>
using namespace std;

int main()
{
    int num;
    num = 22;
    cout << "The value stored in num is " << num << endl;
    cout << "The address of num = " << &num << endl;
    return 0;
}
A Case Study: Radar Speed Trap

• Step 1: Analyze the Problem
  – Understand the desired outputs
  – Determine the required inputs
• Step 2: Develop a Solution
  – Determine the algorithms to be used
  – Use top-down approach to design
• Step 3: Code the Solution
• Step 4: Test and Correct the Program
• **Analyze the Problem**
  – Output: Speed of the car
  – Inputs: Emitted frequency and received frequency

• **Develop a Solution**
  – Algorithm:
    • Assign values to f0 and f1
    • Calculate and display speed
A Case Study: Radar Speed Trap (continued)

• Code the Solution

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

int main()
{
    double speed, fe, fr;

    fe = 2e10;
    fr = 2.0000004e10;

    speed = 6.685e8 * (fr - fe) / (fr + fe);
    cout << "The speed is " << speed << " miles/hour " << endl;

    return 0;
}
```
• Test and Correct the Program
  – Verify that the calculation and displayed value agree with the previous hand calculation
  – Use the program with different values of received frequencies
Common Programming Errors

- Omitting the parentheses after `main()`
- Omitting or incorrectly typing the opening brace, `{`, or the closing brace, `}`, that signifies the start and end of a function body
- Misspelling the name of an object or function
- Forgetting to enclose a string sent to `cout` with quotation marks
- Omitting a semicolon at end of statement
Adding a semicolon at end of `#include` statement

`Missing \n` to indicate new line

Substituting letter O for zero and vice versa

Failing to declare all variables
Common Programming Errors (continued)

- Storing an incorrect data type into a variable
- Attempting to use a variable with no value
- Dividing integer values incorrectly
- Mixing data types in the same expression
• A C++ program consists of one or more modules, called functions, one of which must be called main()
• All C++ statements must be terminated by a semicolon
• Data types include int, float, bool, char
• cout object can be used to display data
• cout object requires the preprocessor command #include <iostream>
• Variables must be declared with their data type
• A variable can be used only after it has been declared
• Variables may be initialized when declared
• Definition statement causes computer to allocate memory for a variable
• `sizeof()` operator yields the amount of storage reserved for a variable