C++ Programming: From Problem Analysis to Program Design, Fifth Edition

Chapter 2: Basic Elements of C++
Objectives

In this chapter, you will:

• Become familiar with the basic components of a C++ program, including functions, special symbols, and identifiers
• Explore simple data types
• Discover how to use arithmetic operators
• Examine how a program evaluates arithmetic expressions
Objectives (cont'd.)

• Learn what an assignment statement is and what it does
• Become familiar with the \texttt{string} data type
• Discover how to input data into memory using input statements
• Become familiar with the use of increment and decrement operators
• Examine ways to output results using output statements
Objectives (cont'd.)

• Learn how to use preprocessor directives and why they are necessary
• Learn how to debug syntax errors
• Explore how to properly structure a program, including using comments to document a program
• Learn how to write a C++ program
Introduction

• **Computer program**
  – Sequence of statements whose objective is to accomplish a task

• **Programming**
  – Process of planning and creating a program
A C++ Program

#include <iostream>
using namespace std;
int main()
{
    int num;
    num = 6;
    cout << "My first C++ program." << endl;
    cout << "The sum of 2 and 3 = " << 5 << endl;
    cout << "7 + 8 = " << 7 + 8 << endl;
    cout << "Num = " << num << endl;
    return 0;
}
The Basics of a C++ Program

- **Function**: collection of statements; when executed, accomplishes something
  - May be predefined or standard

- **Syntax**: rules that specify which statements (instructions) are legal

- **Programming language**: a set of rules, symbols, and special words

- **Semantic rule**: meaning of the instruction
Comments

- Comments are for the reader, not the compiler

- Two types:
  - Single line
    ```cpp
    // This is a C++ program. It prints the sentence:
    // Welcome to C++ Programming.
    ```
  - Multiple line
    ```cpp
    /*
    You can include comments that can occupy several lines.
    */
    ```
Special Symbols

- Special symbols

  +, -, *, /, ., ;  
  ?, <=, !=, ==, >=

C++ Programming: From Problem Analysis to Program Design, Fifth Edition
Reserved Words (Keywords)

- Reserved words, keywords, or word symbols
  - Include:
    - int
    - float
    - double
    - char
    - const
    - void
    - return
Identifiers

• Consist of letters, digits, and the underscore character (_)
• Must begin with a letter or underscore
• C++ is case sensitive
  – NUMBER is not the same as number
• Two predefined identifiers are cout and cin
• Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea
Identifiers (cont'd.)

• Legal identifiers in C++:
  - first
  - conversion
  - payRate

<table>
<thead>
<tr>
<th>Illegal Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>employee Salary</code></td>
<td>There can be no space between <code>employee</code> and <code>Salary</code>.</td>
</tr>
<tr>
<td><code>Hello!</code></td>
<td>The exclamation mark cannot be used in an identifier.</td>
</tr>
<tr>
<td><code>one + two</code></td>
<td>The symbol <code>+</code> cannot be used in an identifier.</td>
</tr>
<tr>
<td><code>2nd</code></td>
<td>An identifier cannot begin with a digit.</td>
</tr>
</tbody>
</table>
Whitespaces

• Every C++ program contains whitespaces
  – Include blanks, tabs, and newline characters
• Used to separate special symbols, reserved words, and identifiers
• Proper utilization of whitespaces is important
  – Can be used to make the program readable
Data Types

• **Data type**: set of values together with a set of operations

• C++ data types fall into three categories:
Simple Data Types

• Three categories of simple data
  – **Integral**: integers (numbers without a decimal)
  – **Floating-point**: decimal numbers
  – **Enumeration type**: user-defined data type
Simple Data Types (cont'd.)

• Integral data types are further classified into nine categories:
  - char, short, int, long, bool
  - unsigned char, unsigned short, unsigned int, unsigned long
Simple Data Types (cont'd.)

TABLE 2-2  Values and Memory Allocation for Three Simple Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Values</th>
<th>Storage (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>-2147483648 to 2147483647</td>
<td>4</td>
</tr>
<tr>
<td>bool</td>
<td>true and false</td>
<td>1</td>
</tr>
<tr>
<td>char</td>
<td>-128 to 127</td>
<td>1</td>
</tr>
</tbody>
</table>

- Different compilers may allow different ranges of values.
Int Data Type

• Examples:
  -6728
  0
  78
  +763

• Positive integers do not need a + sign

• No commas are used within an integer
  – Commas are used for separating items in a list
bool Data Type

• bool type
  – Two values: true and false
  – Manipulate logical (Boolean) expressions
• true and false
  – Logical values
• bool, true, and false
  – Reserved words
char Data Type

• The smallest integral data type
• Used for characters: letters, digits, and special symbols
• Each character is enclosed in single quotes
  – 'A', 'a', '0', '*', '+', '$', '&'
• A blank space is a character
  – Written ' ', with a space left between the single quotes
Floating-Point Data Types

- C++ uses scientific notation to represent real numbers (floating-point notation)

<table>
<thead>
<tr>
<th>Real Number</th>
<th>C++ Floating-Point Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>75.924</td>
<td>7.592400E+1</td>
</tr>
<tr>
<td>0.18</td>
<td>1.800000E-1</td>
</tr>
<tr>
<td>0.00000453</td>
<td>4.530000E-5</td>
</tr>
<tr>
<td>-1.482</td>
<td>-1.482000E0</td>
</tr>
<tr>
<td>7800.0</td>
<td>7.800000E+3</td>
</tr>
</tbody>
</table>
Floating-Point Data Types (cont'd.)

- **float**: represents any real number
  - Range: -3.4E+38 to 3.4E+38 (four bytes)
- **double**: represents any real number
  - Range: -1.7E+308 to 1.7E+308 (eight bytes)
Floating-Point Data Types (cont'd.)

• Maximum number of significant digits (decimal places) for float values is 6 or 7

• Maximum number of significant digits for double is 15

• Precision: maximum number of significant digits
  – Float values are called single precision
  – Double values are called double precision
Arithmetic Operators and Operator Precedence

• C++ arithmetic operators:
  – + addition
  – - subtraction
  – * multiplication
  – / division
  – % modulus operator

• +, -, *, and / can be used with integral and floating-point data types

• Operators can be unary or binary
Order of Precedence

• All operations inside of () are evaluated first
• *, /, and % are at the same level of precedence and are evaluated next
• + and – have the same level of precedence and are evaluated last
• When operators are on the same level
  – Performed from left to right (associativity)

\[3 \times 7 - 6 + 2 \times 5 / 4 + 6\] means
\[(((3 \times 7) - 6) + ((2 \times 5) / 4)) + 6\]
Expressions

• If all operands are integers
  – Expression is called an integral expression
    • Yields an integral result
    • Example: \( 2 + 3 \times 5 \)

• If all operands are floating-point
  – Expression is called a floating-point expression
    • Yields a floating-point result
    • Example: \( 12.8 \times 17.5 - 34.50 \)
Mixed Expressions

• Mixed expression:
  – Has operands of different data types
  – Contains integers and floating-point

• Examples of mixed expressions:
  2 + 3.5
  6 / 4 + 3.9
  5.4 * 2 - 13.6 + 18 / 2
Mixed Expressions (cont'd.)

• Evaluation rules:
  – If operator has same types of operands
    • Evaluated according to the type of the operands
  – If operator has both types of operands
    • Integer is changed to floating-point
    • Operator is evaluated
    • Result is floating-point
  – Entire expression is evaluated according to precedence rules
Type Conversion (Casting)

- **Implicit type coercion**: when value of one type is automatically changed to another type
- **Cast operator**: provides explicit type conversion
  
  ```cpp
  static_cast<dataTypeName>(expression)
  ```
Type Conversion (cont'd.)

**Example 2-9**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Evaluates to</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>static_cast&lt;int&gt;(7.9)</code></td>
<td>7</td>
</tr>
<tr>
<td><code>static_cast&lt;int&gt;(3.3)</code></td>
<td>3</td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(25)</code></td>
<td>25.0</td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(5+3)</code></td>
<td>Equality operator results:</td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(15) / 2</code></td>
<td>7</td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(15 / 2)</code></td>
<td>Equality operator results:</td>
</tr>
<tr>
<td><code>static_cast&lt;int&gt;(7.8 + static_cast&lt;double&gt;(15) / 2)</code></td>
<td>Equality operator results:</td>
</tr>
<tr>
<td><code>static_cast&lt;int&gt;(7.8 + static_cast&lt;double&gt;(15 / 2))</code></td>
<td>Equality operator results:</td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;double&gt;(8) = 8.0</code></td>
</tr>
<tr>
<td></td>
<td>= <code>15.0 / 2</code></td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;double&gt;(15) = 15.0</code></td>
</tr>
<tr>
<td></td>
<td>= <code>15.0 / 2.0 = 7.5</code></td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;double&gt;(7)</code> (because <code>15 / 2 = 7</code>)</td>
</tr>
<tr>
<td></td>
<td>= 7.0</td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;int&gt;(7.8 + 7.5)</code></td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;int&gt;(15.3)</code></td>
</tr>
<tr>
<td></td>
<td>= 15</td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;int&gt;(7.8 + 7.0)</code></td>
</tr>
<tr>
<td></td>
<td>= <code>static_cast&lt;int&gt;(14.8)</code></td>
</tr>
<tr>
<td></td>
<td>= 14</td>
</tr>
</tbody>
</table>
string Type

- Programmer-defined type supplied in ANSI/ISO Standard C++ library
- Sequence of zero or more characters
- Enclosed in double quotation marks
- **Null**: a string with no characters
- Each character has relative position in string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of "William Jacob" is 13
Input

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
  - Instruct computer to allocate memory
  - Include statements to put data into memory
Allocating Memory with Constants and Variables

• **Named constant**: memory location whose content can’t change during execution

• The syntax to declare a named constant is:
  ```
  const dataType identifier = value;
  ```

• In C++, `const` is a reserved word

---

**EXAMPLE 2-11**

Consider the following C++ statements:

```cpp
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
const double PAY_RATE = 15.75;
```
Allocating Memory with Constants and Variables (cont'd.)

- **Variable**: memory location whose content may change during execution

- The syntax to declare a named constant is:

```cpp
dataType identifier, identifier, . . . ;
```

**EXAMPLE 2-12**

Consider the following statements:

```cpp
double amountDue;
int counter;
char ch;
int x, y;
string name;
```
Putting Data into Variables

• Ways to place data into a variable:
  – Use C++’s assignment statement
  – Use input (read) statements
Assignment Statement

• The assignment statement takes the form:

```c
variable = expression;
```

• Expression is evaluated and its value is assigned to the variable on the left side

• In C++, = is called the assignment operator
Assignment Statement (cont'd.)

EXAMPLE 2-13

```c++
int num1, num2;
double sale;
char first;
string str;

num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day."
```

EXAMPLE 2-14

1. num1 = 18;
2. num1 = num1 + 27;
3. num2 = num1;
4. num3 = num2 / 5;
5. num3 = num3 / 4;
Saving and Using the Value of an Expression

• To save the value of an expression:
  – Declare a variable of the appropriate data type
  – Assign the value of the expression to the variable that was declared
    • Use the assignment statement
• Wherever the value of the expression is needed, use the variable holding the value
Declaring & Initializing Variables

• Variables can be initialized when declared:
  
  ```
  int first=13, second=10;
  char ch=' ';
  double x=12.6;
  ```

• All variables must be initialized before they are used
  – But not necessarily during declaration
Input (Read) Statement

• `cin` is used with `>>` to gather input

```
cin >> variable >> variable ...;
```

• The **stream extraction operator** is `>>`
• For example, if `miles` is a double variable
  ```
cin >> miles;
  ```
  – Causes computer to get a value of type `double`
  – Places it in the variable `miles`
Input (Read) Statement (cont'd.)

• Using more than one variable in `cin` allows more than one value to be read at a time.

• For example, if `feet` and `inches` are variables of type `int`, a statement such as:

```cpp
cin >> feet >> inches;
```

– Inputs two integers from the keyboard
– Places them in variables `feet` and `inches` respectively.
EXAMPLE 2-17

```cpp
#include <iostream>

using namespace std;

int main()
{
    int feet;
    int inches;

    cout << "Enter two integers separated by spaces: ";
    cin >> feet >> inches;
    cout << endl;

    cout << "Feet = " << feet << endl;
    cout << "Inches = " << inches << endl;

    return 0;
}
```

**Sample Run:** (In this sample run, the user input is shaded.)

Enter two integers separated by spaces: 23 7

Feet = 23
Inches = 7
Variable Initialization

- There are two ways to initialize a variable:
  ```cpp
  int feet;
  - By using the assignment statement
    feet = 35;
  - By using a read statement
    cin >> feet;
  ```
Increment and Decrement Operators

• Increment operator: increment variable by 1
  – Pre-increment: `++variable`
  – Post-increment: `variable++`

• Decrement operator: decrement variable by 1
  – Pre-decrement: `--variable`
  – Post-decrement: `variable--`

• What is the difference between the following?

\[
\begin{align*}
  x &= 5; \\
  y &= ++x; \\
  x &= 5; \\
  y &= x++; \\
\end{align*}
\]
Output

• The syntax of `cout` and `<<` is:

```cpp
cout << expression or manipulator << expression or manipulator...;
```

– Called an **output statement**

• The **stream insertion operator** is `<<`

• Expression evaluated and its value is printed at the current cursor position on the screen
Output (cont'd.)

• A manipulator is used to format the output
  – Example: \texttt{endl} causes insertion point to move to beginning of next line

\begin{example}
\begin{tabular}{ll}
\textbf{Statement} & \textbf{Output} \\
1 cout \texttt{\llap{<} 29 / 4} \texttt{\llap{<} endl;} & 7 \\
2 cout \texttt{\llap{<} "Hello there."} \texttt{\llap{<} endl;} & Hello there. \\
3 cout \texttt{\llap{<} 12} \texttt{\llap{<} endl;} & 12 \\
4 cout \texttt{\llap{<} "4 + 7"} \texttt{\llap{<} endl;} & 4 + 7 \\
5 cout \texttt{\llap{<} 4 + 7} \texttt{\llap{<} endl;} & 11 \\
6 cout \texttt{\llap{<} 'A'} \texttt{\llap{<} endl;} & A \\
7 cout \texttt{\llap{<} "4 + 7 = "} \texttt{\llap{<} 4 + 7} \texttt{\llap{<} endl;} & 4 + 7 = 11 \\
8 cout \texttt{\llap{<} 2 + 3 * 5} \texttt{\llap{<} endl;} & 17 \\
9 cout \texttt{\llap{<} "Hello \nthere."} \texttt{\llap{<} endl;} & Hello there.
\end{tabular}
\end{example}
Output (cont'd.)

• The new line character is \n
  – May appear anywhere in the string

```cpp
cout << "Hello there.\n";
cout << "My name is James.";  
```  

• Output:
  Hello there. My name is James.

```cpp
cout << "Hello there.\n";
cout << "My name is James.";  
```  

• Output:
  Hello there.
  My name is James.
## TABLE 2-4 Commonly Used Escape Sequences

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Newline</td>
<td>Cursor moves to the beginning of the next line</td>
</tr>
<tr>
<td>\t Tab</td>
<td>Cursor moves to the next tab stop</td>
</tr>
<tr>
<td>\b Backspace</td>
<td>Cursor moves one space to the left</td>
</tr>
<tr>
<td>\r Return</td>
<td>Cursor moves to the beginning of the current line (not the next line)</td>
</tr>
<tr>
<td>\ \ Backslash</td>
<td>Backslash is printed</td>
</tr>
<tr>
<td>' Single quotation</td>
<td>Single quotation mark is printed</td>
</tr>
<tr>
<td>&quot; Double quotation</td>
<td>Double quotation mark is printed</td>
</tr>
</tbody>
</table>
Preprocessor Directives

• C++ has a small number of operations
• Many functions and symbols needed to run a C++ program are provided as collection of libraries
• Every library has a name and is referred to by a header file
• Preprocessor directives are commands supplied to the preprocessor
• All preprocessor commands begin with #
• No semicolon at the end of these commands
Preprocessor Directives (cont'd.)

• Syntax to include a header file:

```cpp
#include <headerFileName>
```

• For example:

```cpp
#include <iostream>
```

– Causes the preprocessor to include the header file `iostream` in the program
namespace and Using cin and cout in a Program

- cin and cout are declared in the header file iostream, but within std namespace

- To use cin and cout in a program, use the following two statements:
  
  ```
  #include <iostream>
  using namespace std;
  ```
Using the `string` Data Type in a Program

• To use the `string` type, you need to access its definition from the header file `string`

• Include the following preprocessor directive:

  ```cpp
  #include <string>
  ```
Creating a C++ Program

• C++ program has two parts:
  – Preprocessor directives
  – The program

• Preprocessor directives and program statements constitute C++ source code (.cpp)

• Compiler generates object code (.obj)

• Executable code is produced and saved in a file with the file extension .exe
Creating a C++ Program (cont'd.)

• A C++ program is a collection of functions, one of which is the function main

• The first line of the function main is called the heading of the function:
  – `int main()`

• The statements enclosed between the curly braces (`{` and `}`) form the body of the function
  – Contains two types of statements:
    • Declaration statements
    • Executable statements
Creating a C++ Program (cont'd.)

**EXAMPLE 2-29**

```cpp
#include <iostream> //Line 1
using namespace std; //Line 2
const int NUMBER = 12; //Line 3

int main() //Line 4
{
    int firstNum; //Line 5
    int secondNum; //Line 6

    firstNum = 18; //Line 8
    cout << "Line 9: firstNum = " << firstNum << endl; //Line 9

    cout << "Line 10: Enter an integer: "; //Line 10
    cin >> secondNum; //Line 11
    cout << endl; //Line 12

    cout << "Line 13: secondNum = " << secondNum << endl; //Line 13

    firstNum = firstNum + NUMBER + 2 * secondNum; //Line 14
    cout << "Line 15: The new value of " << "firstNum = " << firstNum << endl; //Line 15

    return 0; //Line 16
} //Line 17
```
Creating a C++ Program
(cont'd.)

Sample Run:

Line 9: firstNum = 18
Line 10: Enter an integer: 15

Line 13: secondNum = 15
Line 15: The new value of firstNum = 60
Debugging: Understanding and Fixing Syntax Errors

• Compile a program
  – Compiler will identify the syntax error
  – Specifies the line numbers where the errors occur

Example2_Syntax.Errors.cpp
  c:\chapter 2 source code\example2_syntax_errors.cpp(9) : error C2146: syntax error :
  missing ';' before identifier 'num'
  c:\chapter 2 source code\example2_syntax_errors.cpp(11) : error C2065: 'tempNum' :
  undeclared identifier

• Learn how to spot and fix syntax errors
Program Style and Form

• Every C++ program has a function `main`
• Programs must also follow syntax rules
• Other rules serve the purpose of giving precise meaning to the language
Syntax

• Errors in syntax are found in compilation
  
  int x;  //Line 1
  int y   //Line 2: error
  double z;  //Line 3

  y = w + x;  //Line 4: error
Use of Blanks

• In C++, you use one or more blanks to separate numbers when data is input
  – Used to separate reserved words and identifiers from each other and from other symbols
  – Must never appear within a reserved word or identifier
Use of Semicolons, Brackets, and Commas

• All C++ statements end with a semicolon
  – Also called a statement terminator
• { and } are not C++ statements
• Commas separate items in a list
Semantics

• Possible to remove all syntax errors in a program and still not have it run
• Even if it runs, it may still not do what you meant it to do
• For example,
  \[ 2 + 3 \times 5 \text{ and } (2 + 3) \times 5 \]
  are both syntactically correct expressions, but have different meanings
Naming Identifiers

• Identifiers can be **self-documenting**:  
  – CENTIMETERS_PER_INCH

• Avoid **run-together words**:  
  – annualsale
  
  – Solution:
    • Capitalize the beginning of each new word: annualSale
    • Inserting an underscore just before a new word: annual_sale
Prompt Lines

• **Prompt lines**: executable statements that inform the user what to do

```cpp
cout << "Please enter a number between 1 and 10 and "
    << "press the return key" << endl;
cin >> num;
```
Documentation

• A well-documented program is easier to understand and modify
• You use comments to document programs
• Comments should appear in a program to:
  – Explain the purpose of the program
  – Identify who wrote it
  – Explain the purpose of particular statements
Form and Style

• Consider two ways of declaring variables:
  – Method 1
    
    ```
    int feet, inch;
    double x, y;
    ```
  – Method 2
    
    ```
    int feet,inch;double x,y;
    ```

• Both are correct; however, the second is hard to read
More on Assignment Statements

- C++ has special assignment statements called compound assignments $+=, -=, *=, /=, \text{and} \%=$

- Example:

  $$x *= y;$$
Programming Example:
Convert Length

• Write a program that takes as input a given length expressed in feet and inches
  – Convert and output the length in centimeters
• Input: length in feet and inches
• Output: equivalent length in centimeters
• Lengths are given in feet and inches
• Program computes the equivalent length in centimeters
• One inch is equal to $2.54$ centimeters
Programming Example: Convert Length (cont'd.)

• Convert the length in feet and inches to all inches:
  – Multiply the number of feet by 12
  – Add given inches

• Use the conversion formula (1 inch = 2.54 centimeters) to find the equivalent length in centimeters
Programming Example: Convert Length (cont'd.)

• The algorithm is as follows:
  – Get the length in feet and inches
  – Convert the length into total inches
  – Convert total inches into centimeters
  – Output centimeters
Programming Example: Variables and Constants

• Variables

```c++
int feet;      //variable to hold given feet
int inches;    //variable to hold given inches
int totalInches;  //variable to hold total inches
double centimeters;  //variable to hold length in
                    //centimeters
```

• Named Constant

```c++
const double CENTIMETERS_PER_INCH = 2.54;
const int INCHES_PER_FOOT = 12;
```
Programming Example: Main Algorithm

• Prompt user for input
• Get data
• Echo the input (output the input)
• Find length in inches
• Output length in inches
• Convert length to centimeters
• Output length in centimeters
Programming Example: Putting It Together

- Program begins with comments
- System resources will be used for I/O
- Use input statements to get data and output statements to print results
- Data comes from keyboard and the output will display on the screen
- The first statement of the program, after comments, is preprocessor directive to include header file `iostream`
Programming Example: Putting It Together (cont'd.)

• Two types of memory locations for data manipulation:
  – Named constants
    • Usually put before `main`
  – Variables
• This program has only one function (`main`), which will contain all the code
• The program needs variables to manipulate data, which are declared in `main`
Programming Example: Body of the Function

- The body of the function `main` has the following form:

```c
int main ()
{
    declare variables
    statements
    return 0;
}
```
Programming Example: Writing a Complete Program

- Begin the program with comments for documentation
- Include header files
- Declare named constants, if any
- Write the definition of the function `main`
using namespace std;

    // Named constants
const double CENTIMETERS_PER_INCH = 2.54;
const int INCHES_PER FOOT = 12;
int main ()
{

    // Declare variables
    int feet, inches;
    int totalInches;
    double centimeter;

    // Statements: Step 1 - Step 7
    cout << "Enter two integers, one for feet and " << "one for inches: ";
    cin >> feet >> inches;  // Step 1
    cout << endl;
    cout << "The numbers you entered are " << feet << " for feet and " << inches << " for inches. " << endl;  // Step 3
    totalInches = INCHES_PER FOOT * feet + inches;  // Step 4
    cout << "The total number of inches = " << totalInches << endl;  // Step 5
    centimeter = CENTIMETERS_PER_INCH * totalInches;  // Step 6
    cout << "The number of centimeters = " << centimeter << endl;  // Step 7

    return 0;
}
Enter two integers, one for feet, one for inches: 15 7

The numbers you entered are 15 for feet and 7 for inches.
The total number of inches = 187
The number of centimeters = 474.98
Summary

• C++ program: collection of functions where each program has a function called main
• Identifier consists of letters, digits, and underscores, and begins with letter or underscore
• The arithmetic operators in C++ are addition (+), subtraction (-), multiplication (*), division (/), and modulus (%)
• Arithmetic expressions are evaluated using the precedence associativity rules
Summary (cont'd.)

• All operands in an integral expression are integers and all operands in a floating-point expression are decimal numbers
• Mixed expression: contains both integers and decimal numbers
• Use the cast operator to explicitly convert values from one data type to another
• A named constant is initialized when declared
• All variables must be declared before used
Summary (cont'd.)

- Use `cin` and stream extraction operator `>>` to input from the standard input device
- Use `cout` and stream insertion operator `<<` to output to the standard output device
- Preprocessor commands are processed before the program goes through the compiler
- A file containing a C++ program usually ends with the extension `.cpp`