Chapter 2: Basic Elements of C++
Objectives

In this chapter, you will:

- Become familiar with functions, special symbols, and identifiers in C++
- Explore simple data types
- Discover how a program evaluates arithmetic expressions
- Learn about assignment statements
- Become familiar with the string data type
Objectives (cont’d.)

– Learn about assignment statements
– Become familiar with the string data type
– Learn about input and output statements
– Become familiar increment and decrement operators
– Learn how to use preprocessor directives
– Learn how to debug syntax errors
– Explore how to properly structure a program, including using comments to document a program
Introduction

- **Computer program**
  - Sequence of statements whose objective is to accomplish a task
- **Programming**
  - Process of planning and creating a program
- **Real-world analogy: a recipe for cooking**
```cpp
#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
        << "area of a rectangle." << endl;

    length = 6.0;
    width = 4.0;
    perimeter = 2 * (length + width);
    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width = " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```
A C++ Program (cont’d.)

• Sample run:

Program to compute and output the perimeter and area of a rectangle.
Length = 6
Width = 4
Perimeter = 20
Area = 24
A C++ Program (cont’d.)

```cpp
#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and " << endl;
    cout << "area of a rectangle." << endl;
    length = 6.0;
    area = length * width;
    perimeter = 2 * (length + width);
    cout << "Perimeter: " << perimeter << endl;
    cout << "Area: " << area << endl;
}
```

Comments:

- Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it `length`.
- Assignment statement. This statement instructs the system to store 6.0 in the memory space `length`. 
width = 4.0;
perimeter = 2 * (length + width);

area = length * width;

Assignment statement. This statement instructs the system to evaluate the expression length * width and store the result in the memory space area.

cout << "Length = " << length << endl;
cout << "Width = " << width << endl;
cout << "Perimeter = " << perimeter << endl;
cout << "Area = " << area << endl;

return 0;

FIGURE 2-1 Various parts of a C++ program
A C++ Program (cont’d.)

- **Variable**: a memory location whose contents can be changed

```
length  width  area  perimeter
```

**Figure 2-2 Memory allocation**

```
6.0
length  width  area  perimeter
```

**Figure 2-3 Memory spaces after the statement** `length = 6.0;` **executes**
The Basics of a C++ Program

- **Function (or subprogram):** collection of statements; when executed, accomplishes something
  - May be predefined or standard
- **Syntax rules:** rules that specify which statements (instructions) are legal or valid
- **Semantic rules:** determine the meaning of the instructions
- **Programming language:** a set of rules, symbols, and special words
Comments

• Comments are for the reader, not the compiler

• Two types:
  – Single line: begin with //
    // This is a C++ program.
    // Welcome to C++ Programming.
  – Multiple line: enclosed between /* and */
    /*
    You can include comments that can occupy several lines.
    */
Special Symbols

• **Token**: the smallest individual unit of a program written in any language

• C++ tokens include special symbols, word symbols, and identifiers

• Special symbols in C++ include:

```
+   -   *   /   
.   ;   ?   ,   
<=   !=   ==   >=
```
Reserved Words (Keywords)

• **Reserved word symbols** (or **keywords**):
  – Cannot be redefined within program
  – Cannot be used for anything other than their intended use

Examples:
  – int
  – float
  – double
  – char
  – const
  – void
  – return
Identifiers

- **Identifier**: the name of something that appears in a program
  - Consists 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>employee Salary</td>
<td>There can be no space between employee and Salary.</td>
</tr>
<tr>
<td>Hello!</td>
<td>The exclamation mark cannot be used in an identifier.</td>
</tr>
<tr>
<td>one + two</td>
<td>The symbol + cannot be used in an identifier.</td>
</tr>
<tr>
<td>2nd</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 more readable
**Data Types**

- **Data type**: set of values together with a set of operations
- C++ data types fall into three categories:
  - Simple data type
  - Structured data type
  - Pointers
Simple Data Types

• Three categories of simple data
  – **Integral**: integers (numbers without a decimal)
    • Can be further categorized:
      – char, short, int, long, bool, unsigned char, unsigned short, unsigned int, unsigned long
  – **Floating-point**: decimal numbers
  – **Enumeration type**: user-defined data type
Simple Data Types (cont’d.)

<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

• Cannot use a comma within an integer
  – Commas are only 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 single 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
char Data Type (cont’d.)

• Different character data sets exist
• ASCII: American Standard Code for Information Interchange
  – Each of 128 values in ASCII code set represents a different character
  – Characters have a predefined ordering based on the ASCII numeric value
• Collating sequence: ordering of characters based on the character set code
Floating-Point Data Types

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

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>Scientific Notation</th>
<th>C++ Floating-Point Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>75.924</td>
<td>7.5924 * 10^1</td>
<td>7.592400E1</td>
</tr>
<tr>
<td>0.18</td>
<td>1.8 * 10^-1</td>
<td>1.800000E-1</td>
</tr>
<tr>
<td>0.00000453</td>
<td>4.53 * 10^-5</td>
<td>4.530000E-5</td>
</tr>
<tr>
<td>-1.482</td>
<td>-1.482 * 10^0</td>
<td>-1.482000E0</td>
</tr>
<tr>
<td>7800.0</td>
<td>7.8 * 10^3</td>
<td>7.800000E3</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)
- Minimum and maximum values of data types are system dependent
Floating-Point Data Types (cont’d.)

- Maximum number of significant digits (decimal places) for `float` values: 6 or 7
- Maximum number of significant digits for `double`: 15
- **Precision**: maximum number of significant digits
  - Float values are called `single precision`
  - Double values are called `double precision`
Data Types and Variables

• To declare a variable, must specify the data type it will store
  • Syntax: `dataType identifier;`
  • Examples:
    ```
    int counter;
    double interestRate;
    char grade;
    ```
Arithmetic Operators, Operator Precedence, and Expressions

• C++ arithmetic operators:
  – + addition
  – - subtraction
  – * multiplication
  – / division
  – % modulus (or remainder) operator
• +, -, *, and / can be used with integral and floating-point data types
• Use % only with integral data types
• When you use / with integral data types, the integral result is truncated (no rounding)

• **Arithmetic expressions**: contain values and arithmetic operators

• **Operands**: the number of values on which the operators will work

• Operators can be **unary** (one operand) or **binary** (two operands)
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)
• \[ 3 \times 7 - 6 + 2 \times 5 / 4 + 6 \text{ means} \]
  \[ ((3 \times 7) - 6) + ((2 \times 5) / 4)) + 6 \]
Expressions

• **Integral expression**: all operands are integers
  – Yields an integral result
  – Example: \(2 + 3 \times 5\)

• **Floating-point expression**: all operands are floating-point
  – 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:**

\[
2 + 3.5 \\
6 / 4 + 3.9 \\
5.4 * 2 - 13.6 + 18 / 2
\]
• 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
  \[
  \text{static\_cast}<\text{dataTypeName}>(\text{expression})
  \]
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>= <code>static_cast&lt;double&gt;(8) = 8.0</code></td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(15)/2</code></td>
<td>= <code>15.0/2</code></td>
</tr>
<tr>
<td></td>
<td>(because <code>static_cast&lt;double&gt;(15) = 15.0</code>)</td>
</tr>
<tr>
<td><code>static_cast&lt;double&gt;(15/2)</code></td>
<td>= <code>15.0/2.0 = 7.5</code></td>
</tr>
<tr>
<td><code>static_cast&lt;int&gt;(7)</code></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><code>static_cast&lt;int&gt;(7.8 + static_cast&lt;double&gt;(15)/2)</code></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><code>static_cast&lt;int&gt;(7.8 + static_cast&lt;double&gt;(15/2))</code></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** (or **empty**): a string with no characters
- Each character has a relative position in the string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of "William Jacob" is 13
Variables, Assignment Statements, and Input Statements

- 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

- Syntax to declare a named constant:

  ```
  const dataType identifier = value;
  ```

- In C++, `const` is a reserved word

**Example 2-11**

Consider the following C++ statements:

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

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

• Syntax to declare a named constant:

```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:
  \[ \text{variable} = \text{expression}; \]
- Expression is evaluated and its value is assigned to the variable on the left side
- A variable is said to be initialized the first time a value is placed into it
- In C++, = is called the assignment operator
Assignment Statement (cont’d.)

EXAMPLE 2-13

Suppose you have the following variable declarations:

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

Now consider the following assignment statements:

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

• Not all types of variables are initialized automatically
• 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 ...;
```

• This is called an input (read) statement
• The stream extraction operator is `>>`
• For example, if miles is a double variable

```
cin >> miles;
```
  Causes computer to get a value of type `double` and
  places it in the variable `miles`
• Using more than one variable in `cin` allows more than one value to be read at a time
• Example: if `feet` and `inches` are variables of type `int`, this statement:
  ```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 one or more 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 one or more spaces: 23 7
Feet = 23
Inches = 7
```
Increment and Decrement Operators

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

- Decrement operator: decrease variable by 1
  - Pre-decrement: `--variable`
  - Post-decrement: `variable--`

- What is the difference between the following?

```cpp
x = 5;
y = ++x;  // y = 6, x = 6

x = 5;
y = x++; // y = 5, x = 6
```
Output

• The syntax of `cout` and `<<` is:

```
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: `endl` causes insertion point to move to beginning of next line

**EXAMPLE 2-21**

Consider the following statements. The output is shown to the right of each statement.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cout &lt;&lt; 29 / 4 &lt;&lt; endl;</td>
<td>7</td>
</tr>
<tr>
<td>2 cout &lt;&lt; &quot;Hello there.&quot; &lt;&lt; endl;</td>
<td>Hello there.</td>
</tr>
<tr>
<td>3 cout &lt;&lt; 12 &lt;&lt; endl;</td>
<td>12</td>
</tr>
<tr>
<td>4 cout &lt;&lt; &quot;4 + 7&quot; &lt;&lt; endl;</td>
<td>4 + 7</td>
</tr>
<tr>
<td>5 cout &lt;&lt; 4 + 7 &lt;&lt; endl;</td>
<td>11</td>
</tr>
<tr>
<td>6 cout &lt;&lt; 'A' &lt;&lt; endl;</td>
<td>A</td>
</tr>
<tr>
<td>7 cout &lt;&lt; &quot;4 + 7 = &quot; &lt;&lt; 4 + 7 &lt;&lt; endl;</td>
<td>4 + 7 = 11</td>
</tr>
<tr>
<td>8 cout &lt;&lt; 2 + 3 * 5 &lt;&lt; endl;</td>
<td>17</td>
</tr>
</tbody>
</table>
| 9 cout << "Hello 
there." << endl; | Hello there. |
Output (cont’d.)

- The new line character is '\n'
  - May appear anywhere in the string

```cpp
cout << "Hello there."
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>\n</td>
<td>Newline, cursor moves to the beginning of the next line</td>
</tr>
<tr>
<td>\t</td>
<td>Tab, cursor moves to the next tab stop</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace, cursor moves one space to the left</td>
</tr>
<tr>
<td>\r</td>
<td>Return, cursor moves to the beginning of the current line (not the next line)</td>
</tr>
<tr>
<td>\</td>
<td>Backslash is printed</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation mark is printed</td>
</tr>
<tr>
<td>&quot;</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 program
- All preprocessor commands begin with #
- No semicolon at the end of these commands
Preprocessor Directives (cont’d.)

• Syntax to include a header file:

```
#include <headerFileName>
```

• For example:

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

  – Causes the preprocessor to include the header file `iostream` in the program

• Preprocessor commands are processed before the program goes through the compiler
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

• 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
Creating a C++ Program (cont’d.)

• A C++ program contains two types of statements:
  – Declaration statements: declare things, such as variables
  – Executable statements: perform calculations, manipulate data, create output, accept input, etc.
Creating a C++ Program (cont’d.)

• 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
• Compile a program
  – Compiler will identify the syntax errors
  – 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
Syntax

- **Syntax rules**: indicate what is legal and what is not legal
- **Errors in syntax are found in compilation**

```c++
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.
- Blanks are also used to separate reserved words and identifiers from each other and from other symbols.
- Blanks 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
  – Can be regarded as delimiters

• Commas separate items in a list
Semantics

• **Semantics**: set of rules that gives meaning to a language
  – 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

• Ex: \(2 + 3 * 5\) and \((2 + 3) * 5\) 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:
    • Capitalizing 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;
  ```

- **Always include prompt lines when input is needed from users**
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
    \[
    \text{int } \text{feet, inch;}
    \]
    \[
    \text{double } x, y;
    \]
  – Method 2
    \[
    \text{int feet, inch; double } x, y;
    \]
• Both are correct; however, the second is hard to read
Summary

• C++ program: collection of functions, one of which is always called **main**

• Identifiers consist 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
• 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`