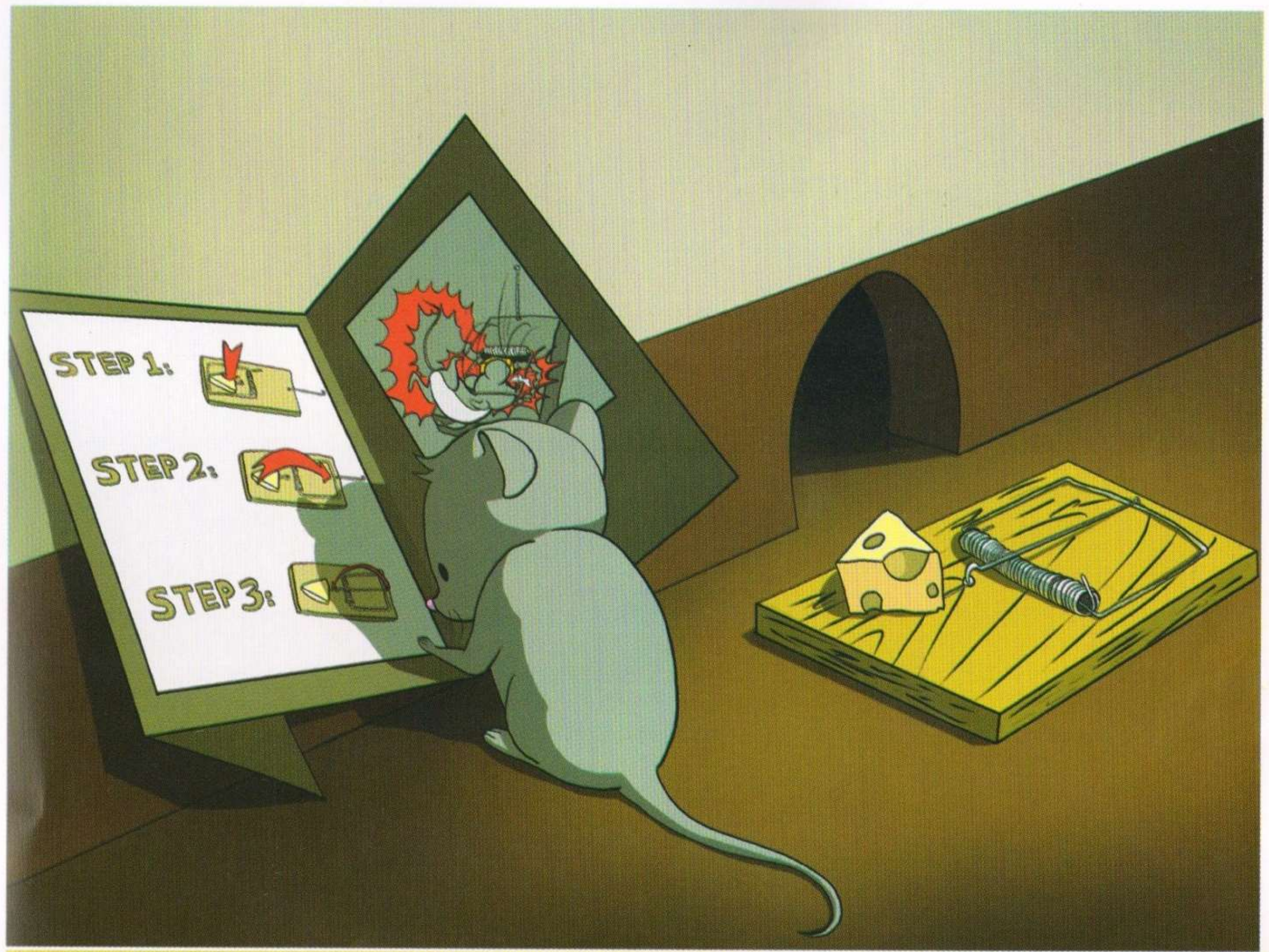
2

Introduction to C++ Programming

杨冠羽
东南大学软件学院



宁可间接的痛苦学习也好过直接的痛苦经历。

The more hard lessons you can learn vicariously rather than through your own hard experience the better.

OBJECTIVES

In this chapter you'll learn:

- To write simple computer programs in C++.
- To write simple input and output statements 输入输出语句.
- To use fundamental types 基本数据类型.
- Basic computer memory concepts 计算机内存概念.
- To use arithmetic operators 算术运算符.
- The precedence of arithmetic operators 运算符的优先级.
- To write simple decision-making statements 简单判断语句.

2.1 Introduction

- C++ programming
  - Programs process information and display results
- Five examples demonstrate
  - How to display messages on the screen
    如何在屏幕上显示信息
  - How to obtain information from the user
    如何从用户获取信息
  - How to perform arithmetic calculations
    如何进行数学计算
  - How to make decisions by comparing numbers
    如何比较两个数

2.2 First Program in C++: Printing a Line of Text

- Simple program
  - Prints a line of text
  - Illustrates several important features of C++

```cpp
// Fig. 2.1: fig02_01.cpp
// Text-printing program.
#include <iostream>

// function main begins program execution
int main()
{
  std::cout << "Welcome to C++!\n";

  return 0;

} // end function main
```

2.2 First Program in C++: Printing a Line of Text (Cont.)

- Comments 注释
  - `// This is a text-printing program.` （Fig2.1 L2）
  - Explain programs to you and other programmers
  - Improve program readability
  - Ignored by compiler
  - Single-line comment 单行注释
    - Begins with //
  - Multi-line comment 多行注释
    - Starts with /*
    - Ends with */

Good Programming Practice 2.1

Every program should begin with a comment that describes the purpose of the program, author, date and time.

2.2 First Program in C++: Printing a Line of Text

- Preprocessor directives 预处理指令
  - `#include <iostream>` （Fig2.1 L3）
  - Processed by preprocessor before compiling (Append F)
  - Begin with #
  - Example
    - Tells preprocessor to include the input/output stream header file `<iostream>`
- White space 白字符
  - Blank lines, space characters and tabs 空行，空字符，制表符（Fig2.1， L4)
  - Used to make programs easier to read
  - Ignored by the compiler 被编译器忽略

2.2 First Program in C++: Printing a Line of Text

- Function `main`
  - `int main() (Fig 2, L6)`
  - A part of every C++ program
    - Exactly one function in a program must be `main`
  - Can return a value
  - Body is delimited by braces (`{}`)
  - Example
    - This `main` function returns an integer (whole number)
- Statements 语句
  - Instruct the program to perform an action
  - All statements end with a semicolon “**;**”

2.2 First Program in C++: Printing a Line of Text

- `std::cout << “Welcome to C++!\n”; (Fig 2.1, L8)`
- Stream insertion operator << 流插入运算符
  - Value to right (right operand) inserted into left operand
  - Example
    - Inserts the string “xxx" into the standard output
      - Displays to the screen
- Escape characters 转义字符
  - A character preceded by "\"
    - Indicates “special” character output
  - Example
    - “\n”: Cursor moves to beginning of next line on the screen

2.2 First Program in C++: Printing a Line of Text

- Namespace 名字空间
  - `std::`(作用域分辨运算符)
    - Specifies using a name that belongs to “namespace” `std`
    - Can be removed through the use of `using` statements
      - using namespace std;
      - using std::cout;
- Standard output stream object 标准输出流对象
  - `std::cout`
    - “Connected” to screen
    - Defined in input/output stream header file `<iostream>`

2.2 First Program in C++: Printing a Line of Text

- `return` statement
  - `return 0;`（Fig2.1 L10）
  - One of several means to exit a function 函数结束的方法之一
  - When used at the end of `main`
    - The value 0 indicates the program terminated successfully
    - 返回值为0，返回给系统

Escape sequence	Description
`\n`	Newline. Position the screen cursor to the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Used to print a backslash character.
`\'`	Single quote. Use to print a single quote character.
`\"`	Double quote. Used to print a double quote character.

Fig. 2.2 | Escape sequences. 转义序列（转义字符）

2.3 Modifying Our First C++ Program

- Two examples
  - Print text on one line using multiple statements (Fig. 2.3)
    - Each stream insertion resumes printing where the previous one stopped
  - Print text on several lines using a single statement (Fig. 2.4)
    - Each newline escape sequence positions the cursor to the beginning of the next line
    - Two newline characters back-to-back outputs a blank line

```cpp
// Fig. 2.3: fig02_03.cpp
// Printing a line of text with multiple statements.
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome ";
   std::cout << "to C++!\n";

   return 0; // indicate that program ended successfully

} // end function main
```

```cpp
// Fig. 2.4: fig02_04.cpp
// Printing multiple lines of text with a single statement.
#include <iostream>

// function main begins program execution
int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";

   return 0; // indicate that program ended successfully

} // end function main
```

2.4 Another C++ Program: Adding Integers

- Problem
  - Adding Integers 两个整数相加
- Pseudocode 伪代码

```
Step 1: Define variables (变量)
Step 2: Prompt the user to input the integers
Step 3: Add the integers
Step 4: Output the results
```

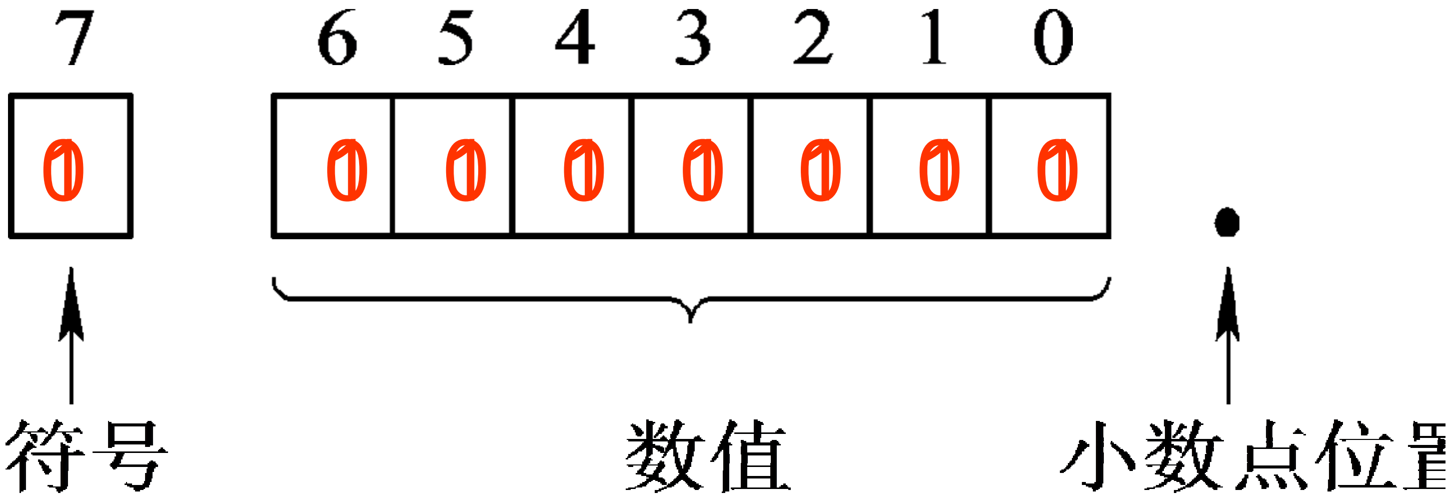
```
#include <iostream>
int main()
{
        int number1;
        int number2;
        int sum;

        std::cout << "Enter first integer: ";
        std::cin >> number1;
        std::cout << "Enter second integer: ";
        std::cin >> number2;
        sum = number1 + number2;
        std::cout << "Sum is " << sum << std::endl;
        return 0;
}
```

2.4 Another C++ Program: Adding Integers

- Variable 变量
  - `int integer1;` （Fig2.5 L9-11）
    `int integer2;`
    `int sum;`
  - Is a location in memory where a value can be stored
  - Common data types (fundamental)
    - `int` – for integer numbers
    - `char` – for characters
    - `double` – for floating point numbers 浮点数
  - Declare variables with data type and name before use 变量声明

定点数（Fixed point numbers)

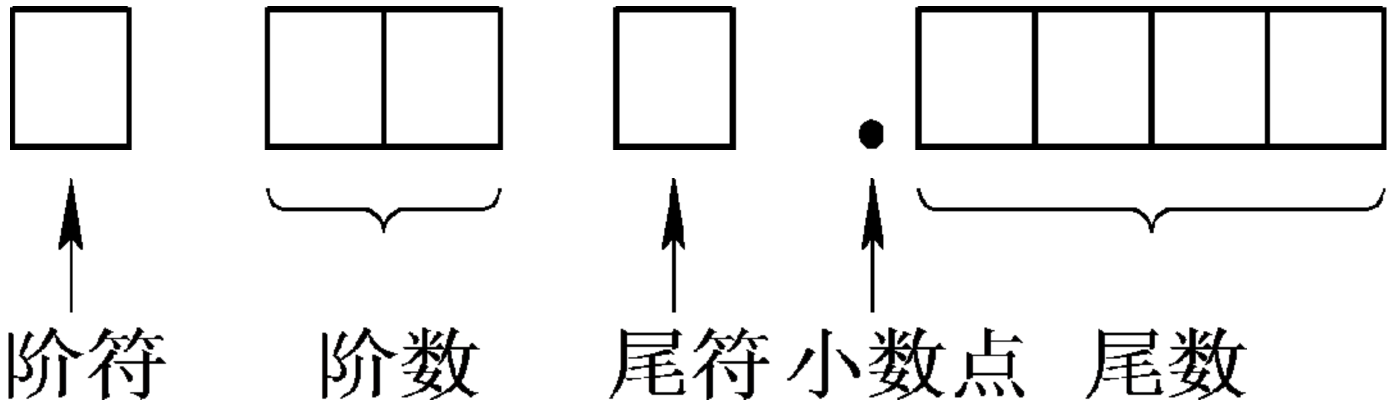


8位定点数的取值范围：

最大值：0111 1111 = 127

最小值：1000 0000 = -128

浮点数（Floating point numbers）



32位浮点数的取值范围：

最大值：3.4E+38

最小值：-3.4E+38

2.4 Another C++ Program: Adding Integers (Cont.)

- Variables (Cont.)
  - You can declare several variables of same type in one declaration
    - Comma-separated list
    - `int integer1, integer2, sum;`
  - Variable name
    - Must be a valid identifier (标识符)
      - Series of characters (letters, digits, underscores 只能包括字母、数字和下划线)
      - Cannot begin with digit 不能以数字作为开头
      - Case sensitive (uppercase letters are *different* from lowercase letters) 大小写敏感

Portability Tip 2.1

C++ allows identifiers of any length, but your C++ implementation may impose some restrictions on the length of identifiers. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice 2.8

Choosing meaningful identifiers helps make a program *self-documenting*—a person can understand the program simply by reading it rather than having to refer to manuals or comments. Avoid using abbreviations in identifiers. This promotes program readability.

For example: F = m * a,

Force = mass * acceleration

Good Programming Practice 2.10

Avoid identifiers that begin with underscores and double underscores, because C++ compilers may use names like that for their own purposes internally. This will prevent names you choose from being confused with names the compilers choose. 避免定义下划线或双下划线开始的标识符。

2.4 Another C++ Program: Adding Integers (Cont.)

- Input stream object 输入流对象
  - `std::cin >> number1; (Fig2.5 L14)`
  - `std::cin` from `<iostream>`
    - Usually connected to keyboard
    - Stream extraction operator `>>` 流提取运算符
      - Waits for user to input value, press *Enter* (*Return*) key
      - Stores a value in the variable to the right of the operator
        - » Converts the value to the variable’s data type
    - Example
        - » Reads an integer typed at the keyboard
        - » Stores the integer in variable `number1`

2.4 Another C++ Program: Adding Integers (Cont.)

- Assignment operator = 赋值运算符
  - `sum = variable1 + variable2;` （Fig2.5， L19）
  - Assigns the value on the right to the variable on the left
  - Binary operator (two operands) 双目运算符（两个操作数）
  - Example:
    - Adds the values of `variable1` and `variable2`
    - Stores the result in the variable `sum`
- Stream manipulator 流运算符
  - `std::endl` （Fig2.5, L21)
  - Outputs a newline
  - Flushes the output buffer 强制刷新输出缓存

2.4 Another C++ Program: Adding Integers (Cont.)

- Concatenating stream insertion operations
  - Use multiple stream insertion operators in a single statement
    - Stream insertion operation knows how to output each type of data
  - Also called chaining or cascading (串联)
  - Example
    - `std::cout << "Sum is " << number1 + number2<< std::endl;`
      - Outputs `"Sum is “`
      - Then outputs the sum of variables `number1` and `number2`
      - Then outputs a newline and flushes the output buffer

2.5 Memory Concepts

- Variable names 变量名称
  - Correspond to actual locations in the computer's memory
    - Every variable has a name, a type, a size and a value （4个要素）
  - When a new value placed into a variable, the new value overwrites the old value
    - Writing to memory is “destructive”（写内存是破坏性的）
  - Reading variables from memory is nondestructive
  - Example
    - `sum = number1 + number2;`
      - Although the value of `sum` *is overwritten*
      - The values of `number1` and `number2` *remain intact*

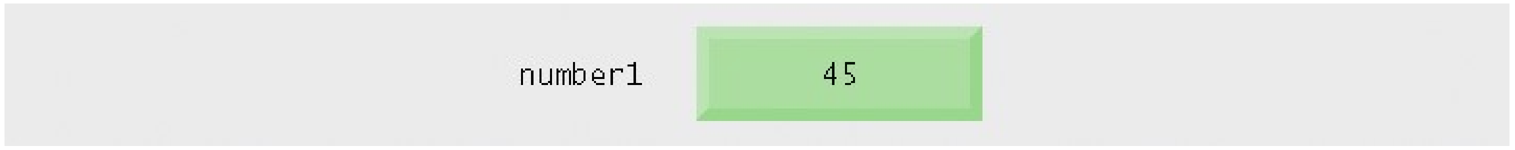


Fig. 2.6 Memory location showing the name and value of variable `number1`.

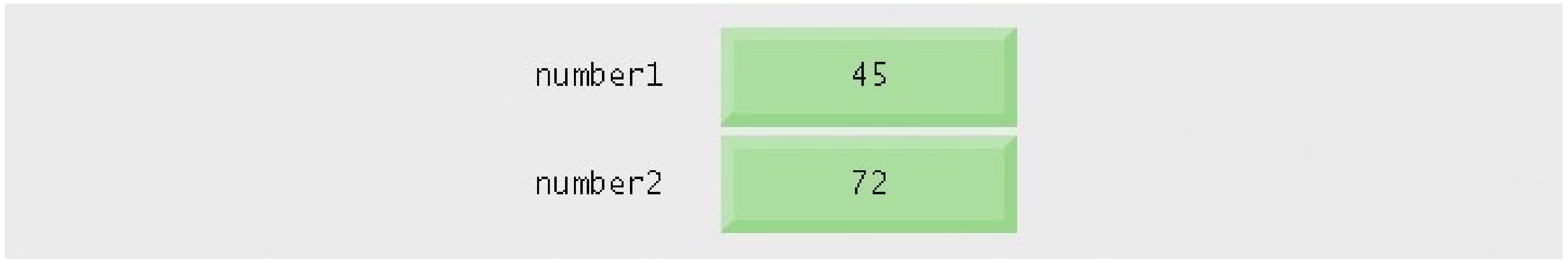


Fig. 2.7 Memory locations after storing values for `number1` and `number2`.

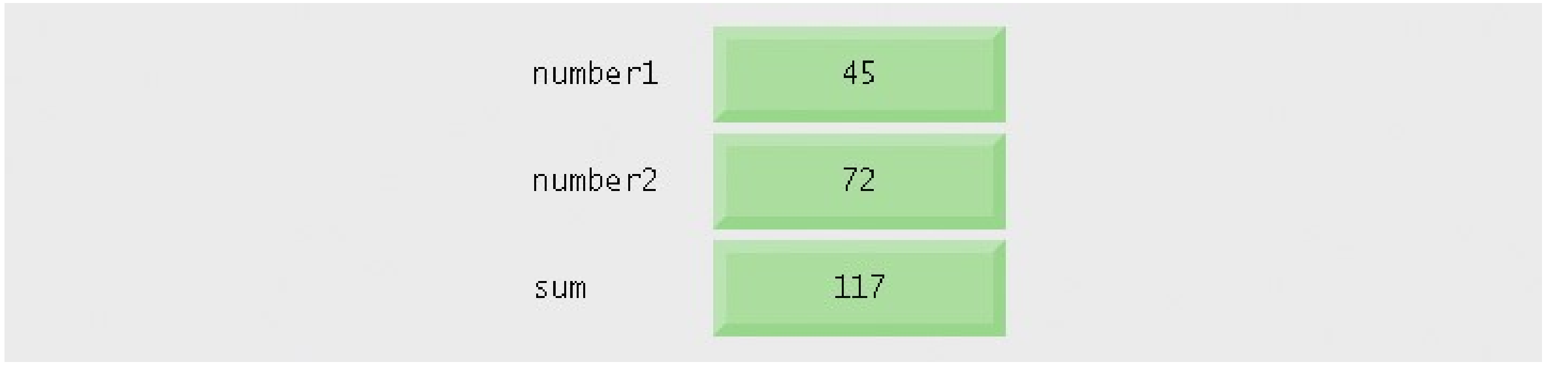


Fig. 2.8 Memory locations after calculating and storing the `sum` of `number1` and `number2`.

2.6 Arithmetic

- Arithmetic operators 算术运算符
  - * 乘法
    - Multiplication
  - / 除法
    - Division
    - Integer division truncates (discards) the remainder
      - 7 / 5 evaluates to 1
  - % 求模（取余数）
    - The modulus operator returns the remainder
      - 7 % 5 evaluates to 2

2.6 Arithmetic (Cont.)

- Straight-line form 直线式
  - Required for arithmetic expressions in C++
  - All constants, variables and operators appear in a straight line (a / b)
- Grouping subexpressions
  - Parentheses 括号 are used in C++ expressions to group subexpressions
    - In the same manner as in algebraic expressions
  - Example
    - `a * ( b + c )`
      - Multiple `a` times the quantity `b + c`

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	*bm* or $b \cdot m$	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	`%`	*r mod s*	`r % s`

Fig. 2.9 | Arithmetic operators.

2.6 Arithmetic (Cont.)

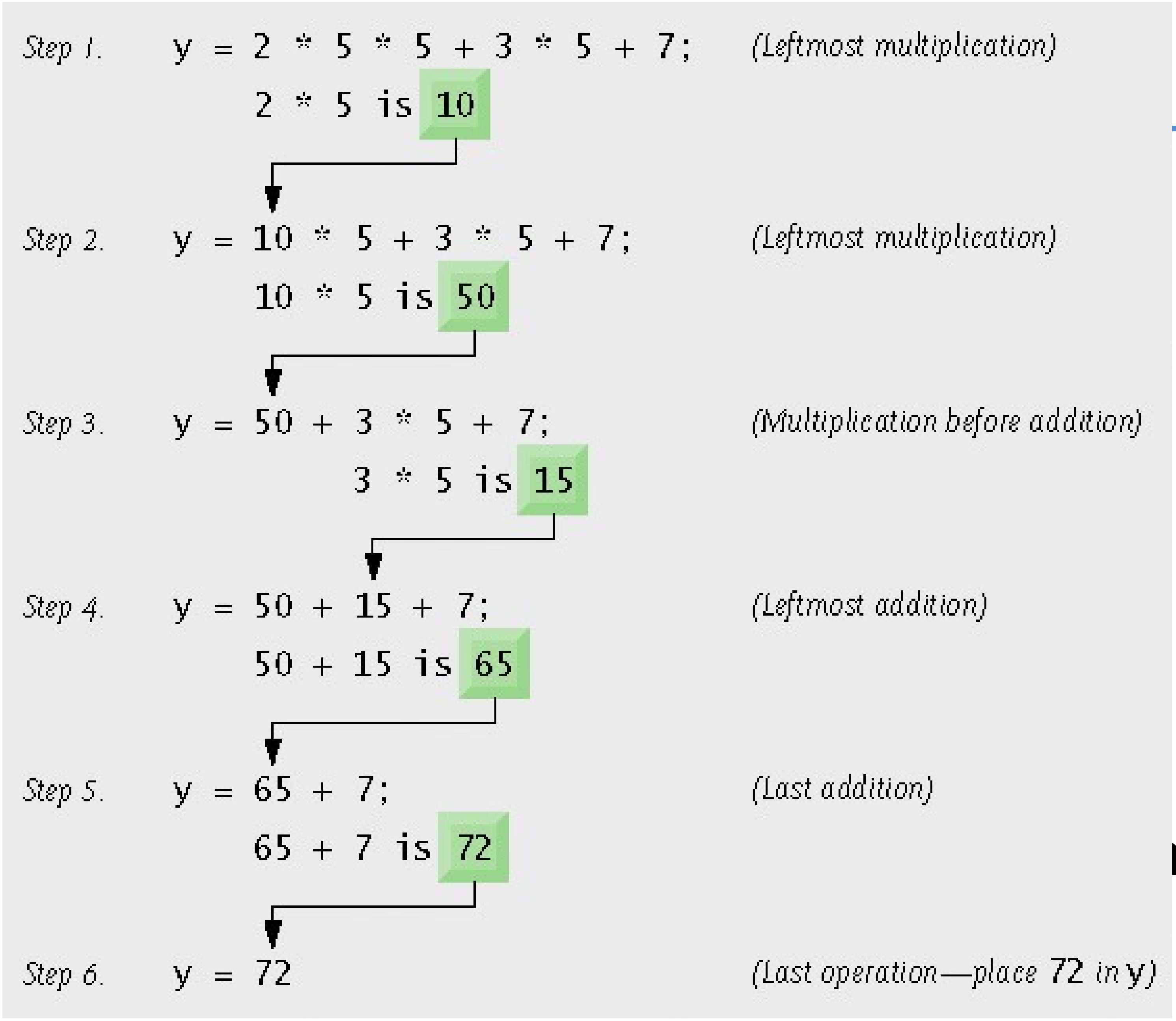
- Rules of operator precedence 运算符优先级规则
  - Operators in parentheses (括号) are evaluated first
    - For nested (embedded) parentheses
      - Operators in innermost pair are evaluated first
  - Multiplication, division and modulus are applied next
    - Operators are applied from left to right
  - Addition and subtraction are applied last
    - Operators are applied from left to right

括号内最先，乘除取余数其次，最后加减法

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
* / %	Multiplication Division Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Fig. 2.10 | **Precedence of arithmetic operators.** 运算符优先级

Append. A 运算符优先级表



```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

Fig. 2.11

nial

2.7 Decision Making: Equality and Relational Operators 判断：相等或关系运算符

- Condition 条件
  - Expression can be either `true` or `false` （取值非真即假）
  - Can be formed using equality or relational operators 采用相等运算符或关系运算符
- `if` statement if 语句
  - If the condition is `true`, the body of the `if` statement executes
  - If the condition is `false`, the body of the `if` statement does not execute

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	`>`	`x > y`	`x` is greater than `y`
<	`<`	`x < y`	`x` is less than `y`
≥	`>=`	`x >= y`	`x` is greater than or equal to `y`
≤	`<=`	`x <= y`	`x` is less than or equal to `y`
Equality operators			
=	`==`	`x == y`	`x` is equal to `y`
≠	`!=`	`x != y`	`x` is not equal to `y`

Fig. 2.12 | Equality and relational operators.

Common Programming Error 2.5

A syntax error will occur if any of the operators ==, ! =, >= and <= appears with spaces between its pair of symbols.

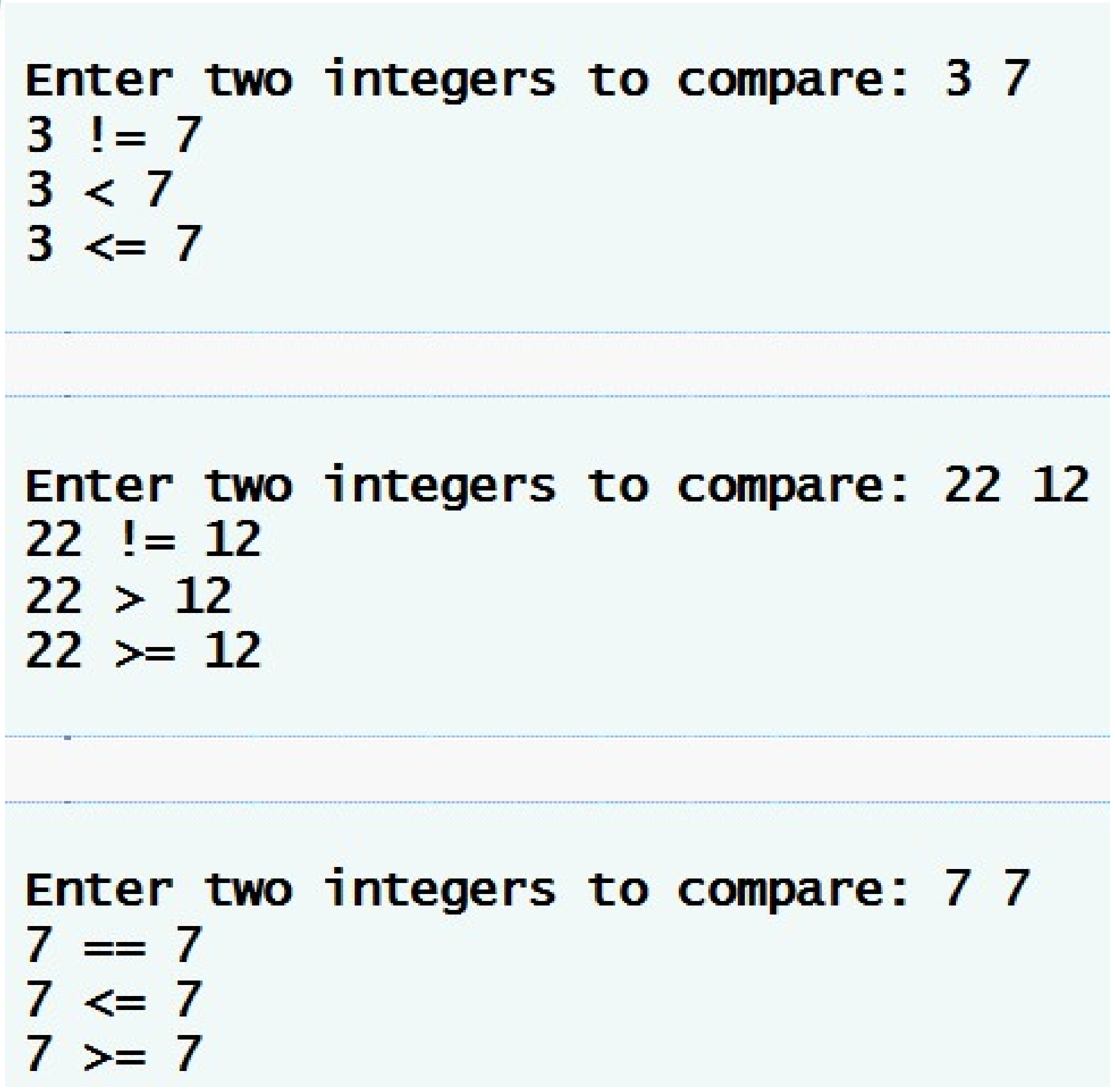
Common Programming Error 2.6

You will understand why when you learn about logical operators in Chapter 5. A *fatal logic error* causes a program to fail and terminate prematurely. A *nonfatal logic error* allows a program to continue executing, but usually produces incorrect results.

```
int main()
{
   int number1;
   int number2;

   cout << "Enter two integers to compare: ";
   cin >> number1 >> number2;

   if ( number1 == number2 )
      cout << number1 << " == " << number2 << endl;
   if ( number1 != number2 )
      cout << number1 << " != " << number2 << endl;
   if ( number1 < number2 )
      cout << number1 << " < " << number2 << endl;
   if ( number1 > number2 )
      cout << number1 << " > " << number2 << endl;
   if ( number1 <= number2 )
      cout << number1 << " <= " << number2 << endl;
   if ( number1 >= number2 )
      cout << number1 << " >= " << number2 << endl;
   return 0;
}
```



```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

Common Programming Error 2.8

Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error).

Example:

```
if ( number1 >= number2 );
        cout << number1 << " >= " << number2 << endl;
```

Operators	Associativity	Type
`()`	left to right	parentheses
`*` `/` `%`	left to right	multiplicative
`+` `-`	left to right	additive
`<<` `>>`	left to right	stream insertion/extraction
`<` `<=` `>` `>=`	left to right	relational
`==` `!=`	left to right	equality
`=`	right to left	assignment

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

Good Programming Practice 2.19

Confirm that the operators in the expression are performed in the order you expect. If you are uncertain about the order of evaluation in a complex expression, break the expression into smaller statements or use parentheses to force the order of evaluation