Objectives

In this chapter, you will learn about:

- Function and parameter declarations 函數和參數宣告
- Returning a single value 回傳單一值
- Returning multiple values 回傳多值
- Rectangular to polar coordinate conversion 直角座標到極座標轉換
- Variable scope 變數有效範圍
- Variable storage categories 變數儲存種類
- Common programming errors 常見錯誤
6.1 function and parameter declarations

函數和參數宣告
Function and Parameter Declarations 函数和参数宣告

• Interaction with a function includes: 使用函数包括
  – Passing data to a function correctly when its called 正确地传资料给函数
  – Returning values from a function when it ceases operation 函数终止时接收回传资料

• A function is called by giving the function’s name and passing arguments in the parentheses following the function name 呼叫函数的方法：函数名稱(參數1、參數2…)
Function and Parameter Declarations (continued)

Program 6.1

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

void findMax(int, int); // the function declaration (prototype)

int main()
{
    int firstnum, secnum;

    cout << "Enter a number: ";
    cin >> firstnum;
    cout << "Great! Please enter a second number: ";
    cin >> secnum;

    findMax(firstnum, secnum); // the function is called here

    return 0;
}
```
• Before a function is called, it must be declared to function that will do calling 函數被呼叫之前要先宣告
• Declaration statement for a function is referred to as function prototype 這個宣告稱為函數原型
• Function prototype tells calling function: 函數原型告訴呼叫函數：
  – Type of value that will be formally returned, if any 回傳資料型態
  – Data type and order of the values the calling function should transmit to the called function 所需提供資料種類和順序
• Function prototypes can be placed with the variable declaration statements above the calling function name or in a separate header file 函數原型可以放在變數宣告處(在呼叫函數之前)或另一個標頭檔案中
Calling a Function 呼叫函數

• Requirements when calling a function include:

  – Using the name of the function 函數名稱
  – Enclosing any data passed to the function in the parentheses following the function name, using the same order and type declared in the function prototype 所需資料或參數放在括號中，順序和種類必須與函數原型相同
Calling a Function (continued)

• The items enclosed in the parentheses are called arguments of the called function. 括號中的參數稱為引數

```c++
findMax (firstnum, secnum);
```

This identifies the `findMax()` function. This causes two values to be passed to `findMax()`.

**Figure 6.2** Calling and passing two values to `findMax()` 呼叫並傳送兩個資料給`findMax()`
Calling a Function (continued)

Figure 6.3 The `findMax()` function receives actual values.

Only parameter values are copied to the function, the original values are not affected.
Calling a Function (continued)

- Figure 6.3:
  - The `findMax()` function does not receive the variables named `firstnum` and `secnum` and has no knowledge of the variable names. 被呼叫函數不知道原始變數名稱的存在
  - The function receives the values in these variables and must then determine where to store those values before it does anything else. 被呼叫函數只是接受數值而且決定該把這些數值放在哪裡
Defining a Function 定義函數

• Every C++ function consists of two parts: 每個C++函數都有兩個部分
  • Function header 函數標頭
  • Function body 函數主體

• Function header’s purpose: 函數標頭的目的
  – Identify data type of value function returns, provide function with name, and specify number, order, and type of arguments function expects 指定函數回傳的資料型態、提供函數名稱、指定引數的數目、順序和型態

• Function body’s purpose: 函數主體的目的
  – To operate on passed data and return, at most, one value directly back to the calling function 將傳來的資料加以運算，而且回傳至多一個結果數值
Defining a Function (continued)

Figure 6.4 The general format of a function

```
function header line
{
    constant and
    variable declarations;
    any other C++ statements;
}
```

Function header

Function body

Figure 6.5 Storing values in parameters

```
findMax(int x, int y)
```

The parameter named x

The parameter named y

此陈述呼叫 findMax( )

The value in firstnum is passed

The value in secnum is passed

函數自己為接收的數值取名字

原始的變數名稱
Program 6.2

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

void findMax(int, int); // the function prototype

int main()
{

    int firstnum, secnum;

    cout << "Enter a number: ";
    cin >> firstnum;
    cout << "Great! Please enter a second number: ";
    cin >> secnum;

    findMax(firstnum, secnum); // the function is called here

    return 0;
}

// following is the function find_max()

void findMax(int x, int y)
{
    int maxnum; // start of function body

    if (x >= y) // find the maximum number
        maxnum = x;
    else
        maxnum = y;

    cout << "The maximum of the two numbers is " << maxnum << endl;

    return; // end of function body and end of function
```
• General rule for placing statements in a C++ program:
  一般規則
  – All preprocessor directives, named constants, variables, and functions must be declared or defined before they can be used 預處理指令、常數、變數和函數可以在任何地方宣告（只要在使用前）
  – Although this rule permits placing both preprocessor directives and declaration statements throughout the program, doing so results in poor program structure 但是這是不佳的程式習慣
Functions with Empty Parameter Lists

- Although useful functions having an empty parameter list are extremely limited, they can occur.

- Function prototype for such a function requires writing the keyword `void` or `nothing` at all between the parenthesis following the function's name.
Default Arguments 預設引數

• C++ provides **default arguments** in a function call for added flexibility 在C++函數呼叫中可以有預設引數

• Primary use of default arguments is to extend parameter list of existing functions without requiring any change in calling parameter lists already used in a program 用來延伸引數清單而不必修改程式

• Default arguments are listed in the **function prototype** and transmitted automatically to the called function when the corresponding arguments are omitted from the function call 當對應的引數省略時，函數原型中所設定的預設值會自動傳給函數
Default Arguments (continued)

- **Example: Function prototype with default arguments**
  
  函數原型加上預設引數
  
  ```c++
  void example(int, int = 5, double = 6.78)
  ```

- **Sample valid calls to the example function**
  
  ```c++
  example(7, 2, 9.3) // no defaults used
  example(7, 2)     // same as example(7, 2, 6.78)
  example(7)       // same as example(7, 5, 6.78)
  ```

  若要省略參數從最右方開始省略，不可以只省略中間
Reusing Function Names (Overloading) 函数超载

• C++ provides the capability of using the same function name for more than one function 使用相同的函数名稱
  — Referred to as function overloading 函数超载

• Only requirement for creating more than one function with same name: 唯一要求
  — Compiler must be able determine which function to use based on the parameters’ data types (not the data type of the return value, if any) 编译器必须能根据引数资料型態（必须不同）决定呼叫的函数，而不是根据回傳资料型態
Reusing Function Names (Overloading) (continued)

- Which of the functions is called depends on the argument type supplied at the time of the call. 依照呼叫時提供的資料型態決定哪一個函數被呼叫

```cpp
void cdabs(float x) // compute and display the absolute value of a float
{
    if (x < 0)
        x = -x;
    cout << "The absolute value of the float is " << x << endl;
}

void cdabs(double x) //
    compute and display the absolute value of a double
{
    if (x < 0)
        x = -x;
    cout << "The absolute value of the double is " << x << endl;
}
```
Function Templates 函數模板

• Function template: Single complete function that serves as a model for a family of functions 單一函數作為一群函數的模型
  — Function from the family that is actually created depends on the specific function call 視特定的呼叫產生特定的函數

• Generalize the writing of functions that perform essentially the same operation, but on different parameter data types 便於撰寫使用不同的引數資料種類以處理相同工作的一群函數

• Make it possible to write a general function that handles all cases but where the compiler can set parameters, variables, and even return type based on the actual function call 用來編寫處理各類資料的通用函數，由編譯器依需求選擇資料種類
Program 6.3

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

// Template function to calculate absolute value
template <class T>
void showabs(T number) {
    if (number < 0)
        number = -number;
    cout << "The absolute value of the number is " << number << endl;
    return;
}

int main() {
    int num1 = -4;
    float num2 = -4.23f;
    double num3 = -4.23456;

    showabs(num1);
    showabs(num2);
    showabs(num3);

    return 0;
}
```
#include <iostream>

using namespace std;

// template prefix

// header line

template <class T>
T abs(T value) // variable declaration

{
    T absnum;
    // variable declaration

    if (value < 0)
        absnum = -value;
    else
        absnum = value;

    return (absnum);
}

int main()

{
    int num1 = -4;
    float num2 = -4.23f;
    double num3 = -4.23456;

    cout << "The absolute value of " << num1 << " is " << abs(num1) << endl;
    cout << "The absolute value of " << num2 << " is " << abs(num2) << endl;
    cout << "The absolute value of " << num3 << " is " << abs(num3) << endl;

    return 0;
}
6.2 returning a single value

只回傳單一值的函數
Returning a Single Value 回傳單一值

- Function receiving an argument passed by value cannot inadvertently alter value stored in the variable used for the argument 不會不小心變更原始變數資料

- Function receiving passed by value arguments can process the values sent to it in any fashion and return one, and only one, “legitimate” value directly to the calling function 函數可以處理傳來的資料但最多只能回傳一個值

Figure 6.6 A function directly returns at most one value
Returning a Single Value (continued)

- **findmax function**

```cpp
int findMax(int x, int y) // function header
{
    int maxnum; // variable declaration
    if (x >= y)
        maxnum = x;
    else
        maxnum = y;

    return maxnum; // return statement
}
```

前後資料型態要相同
Inline Functions 行中函數

- Calling a function places a certain amount of overhead on a computer 呼叫函數對電腦造成負擔
  - Placing argument values in a reserved memory region (called the stack) that the function has access to 把引數放在堆疊區
  - Passing control to the function 把控制傳給函數
  - Providing a reserved memory location for any returned value (again, using the stack for this purpose) 將回傳值放在堆疊區中
  - Returning to the correct point in the calling program 將控制交回呼叫者
Inline Functions (continued)

- Paying overhead associated with calling a function is justified when a function is called many times. 如果呼叫該函數很多次是值得的
  - Can reduce a program’s size substantially. 可以減少程式的大小
- For small functions that are not called many times, overhead of passing and returning values might not be warranted. 對於呼叫次數少的小程式則不值得
- **Inline functions:**
  - Group repeating lines of code together under a common function name. 將重複的幾行程式放在一起
  - Have the compiler place this code in the program wherever the function is called. 讓編譯器將程式碼插入
Program 6.5 findMax 函数

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

int findMax(int, int); // the function prototype

int main()
{
    int firstnum, secnum, max;

    cout << "Enter a number: ";
cin >> firstnum;
cout << "Great! Please enter a second number: ";
cin >> secnum;

    max = findMax(firstnum, secnum); // the function is called here
    cout << "The maximum of the two numbers is " << max << endl;
    return 0;
}

int findMax(int x, int y)
{
    int maxnum; // variable declaration
    if (x >= y) // find the maximum number
        maxnum = x;
    else
        maxnum = y;
    return maxnum; // return statement
}
```
#include <iostream>
using namespace std;

double tempvert(double); // function prototype

int main()
{
    const CONVERTS = 4; // number of conversions to be made
    int count;
    double fahrenheit;

    for(count = 1; count <= CONVERTS; count++)
    {
        cout << "Enter a Fahrenheit temperature: ";
        cin >> fahrenheit;
        cout << "The Celsius equivalent is " << tempvert(fahrenheit) << endl;
    }

    return 0;
}

// convert fahrenheit to celsius
double tempvert(double inTemp)
{
    return (5.0/9.0) * (inTemp - 32.0);
}
改用`inline`做溫度轉換

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

// an inline function
inline double tempCtoF(double inTemp) { return (5.0/9.0) * (inTemp - 32.0); }

int main() {
    const CONVERTS = 4; // number of conversions to be made
    int count;
    double fahrenheit;

    for(count = 1; count <= CONVERTS; count++) {
        cout << "Enter a Fahrenheit temperature: ";
        cin >> fahrenheit;
        cout << "The Celsius equivalent is "
            << tempCtoF(fahrenheit) << endl;
    }

    return 0;
}
```
Inline Functions (continued)

- Advantage: Increase in execution speed 行中函數加快執行速度
  - Because the inline function is expanded and included in every expression or statement calling it, no execution time is lost because of the call and return overhead a non-inline function requires 因為是插入程式碼，所以沒有執行時的額外負擔
- Each time an inline function is referenced the complete code is reproduced and stored as an integral part of the program 每次呼叫時，行中函數便會複製一次存入程式中
- A non-inline function is stored in memory only once 一般函數只儲存一次
- Inline functions should be used only for small functions that aren’t called extensively in a program 只應用於不常用的小函數
6.3 returning multiple values

回傳多值的函數
Returning Multiple Values

• In typical function invocation, called function receives values from its calling function stores and manipulates the passed values, and directly returns at most one value.

• Pass by value: When data is passed in this manner.

• Pass by reference: Giving a called function direct access to its calling function’s variables is referred to as.

• The called function can reference, or access, the variable whose address has been passed as a pass by reference argument.
Passing and Using Reference Parameters 傳送和使用參考參數

• From the sending side, calling a function and passing an address as an argument that’s accepted as a reference parameter is the same as calling a function and passing a value. 對呼叫函數的一方來說，傳值和傳地址是相同的工作。

• Whether a value or an address is actually passed depends on the parameter types declared. 傳值或傳地址是依照引數型態而決定。
Figure 6.7 The equivalence of arguments and parameters in Program 6.8
```cpp
#include <iostream>
using namespace std;

void newval(double&, double&); // prototype with two reference parameters

int main()
{
    double firstname, secondname;

    cout << "Enter two numbers: ";
    cin >> firstname >> secondname;
    cout << "\nThe value in firstname is: " << firstname << endl;
    cout << "The value in secondname is: " << secondname << "\n")

    newval(firstname, secondname); // call the function

    cout << "The value in firstname is now: " << firstname << endl;
    cout << "The value in secondname is now: " << secondname << endl;

    return 0;
}

void newval(double& xnum, double& ynum)
{
    cout << "The value in xnum is: " << xnum << endl;
    cout << "The value in ynum is: " << ynum << "\n")
    xnum = 89.5;
    ynum = 99.5;

    return;
}
```
Passing and Using Reference Parameters (continued)

Program 6.9

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

void calc(double, double, double, double&, double&);  // prototype

int main()
{
    double firstnum, seconum, thirdnum, sum, product;

    cout << "Enter three numbers: ";
    cin >> firstnum >> seconum >> thirdnum;
    calc(firstnum, seconum, thirdnum, sum, product);  // function call
    cout << "The sum of the numbers is: " << sum << endl;
    cout << "The product of the numbers is: " << product << endl;

    return 0;
}

void calc(double num1, double num2, double num3, double& total, double& product)
{
    total = num1 + num2 + num3;
    product = num1 * num2 * num3;
    return;
}
```
Program 6.10

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

void swap(double&, double&);  // function receives 2 references

int main()
{
    double firstnum = 20.5, secnum = 6.25;

    cout << "The value stored in firstnum is: " << firstnum << endl;
    cout << "The value stored in secnum is: " << secnum << endl;

    swap(firstnum, secnum);  // call the function with references

    cout << "The value stored in firstnum is now: "
        << firstnum << endl;
    cout << "The value stored in secnum is now: "
        << secnum << endl;

    return 0;
}

void swap(double& num1, double& num2)
{
    double temp;
    temp = num1;  // save num1's value
    num1 = num2;  // store num2's value in num1
    num2 = temp;  // change num2's value

    return;
}
```
Figure 6.8 The relationship between argument and parameter names
Passing and Using Reference Parameters (continued)

Figure 6.9a  Save the first value

Figure 6.9b  Replace the first value with the second value

Figure 6.9c  Change the second value
6.4 a case study
A Case Study: Rectangular to Polar Coordinate Conversion 直角座標轉極座標

Figure 6.10 The problem-solver algorithm
A Case Study: Rectangular to Polar Coordinate Conversion (continued)

Figure 6.11  The correspondence between polar (distance and angle) and Cartesian (x and y) coordinates
Figure 6.12  A top level structure diagram
A Case Study: Rectangular to Polar Coordinate Conversion (continued)

Figure 6.13  Parameter values when `polar()` is called
Program 6.11 程式分解：轉換座標的函數

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

void polar(double, double, double& r, double& theta); // function prototype

int main()
{
    double distance, angle;
    polar(3.0, 4.0, distance, angle);
    cout << "r = " << distance << endl;
    cout << "angle = " << angle << endl;

    return 0;
}

void polar(double x, double y, double& r, double& theta)
{
    const double TODEGREES = 180.0/3.141593;
    r = sqrt(x * x + y * y);
    theta = atan(y/x) * TODEGREES;

    return;
}
```
Program 6.12 程式分解：取得輸入座標的函數

```cpp
#include <iostream>

using namespace std;

void getrec(double&, double&); // function prototype

int main()
{
    double xcoord, ycoord;
    getrec(xcoord, ycoord);
    cout << "The entered value for x is " << xcoord << endl;
    cout << "The entered value for y is " << ycoord << endl;

    return 0;
}

void getrec(double& x, double& y)
{
    cout << "Rectangular to Polar Coordinate"
        << " Conversion Program\n" << endl;
    cout << "Enter the x coordinate: ";
    cin >> x;
    cout << "Enter the y coordinate: ";
    cin >> y;

    return;
}
```
程式分解：顯示結果的函數

Program 6.13
#include <iostream>
using namespace std;

void showit(double, double); // function prototype

int main()
{
    showit(5.0, 53.1301);

    return 0;
}

void showit(double radius, double angle)
{
    cout << "\nThe polar coordinates are: " << endl;
    cout << " Distance from origin: " << radius << endl;
    cout << " Angle (in degrees) from x-axis: " << angle << endl;

    return;
}
Program 6.14

合并三个函数

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

void getrec(double x, double y);    // function prototype
void polar(double x, double y, double r, double theta);
void showit(double radius, double angle);

int main()
{
    double x, y, distance, angle;
    getrec(x, y);
    polar(x, y, distance, angle);
    showit(distance, angle);
    return 0;
}

void getrec(double x, double y)
{
    cout << "Rectangular to Polar Coordinate"
        " Conversion Program\n" << endl;
    cout << "Enter the x coordinate: ";
    cin >> x;
    cout << "Enter the y coordinate: ";
    cin >> y;
    return;
}

void polar(double x, double y, double r, double theta)
{
    const double TODEGREES = 180.0/3.141592;
    r = sqrt(x * x + y * y);
    theta = atan(y/x) * TODEGREES;
    return;
}

void showit(double radius, double angle)
{
    cout << "The polar coordinates are: " << endl;
    cout << "Distance from origin: " << radius << endl;
    cout << "Angle (in degrees) from x-axis: " << angle << endl;
    return;
}
```
6.5 variable scope

變數的有效範圍
• A function can be thought of as a closed box, with slots at the top to receive values and a single slot at the bottom to return a value.

Figure 6.14  A function can be considered a closed box.
Variable Scope (continued)

- **Local variables**: Variables created in a function that are conventionally available only to the function 區域變數：在函數中宣告，只在函數中有效

- **Scope**: Section of the program where the identifier is valid or "known" 有效範圍：識別字可以被辨識的範圍

- A variable with **local scope** is simply one with storage locations set aside for it by a declaration statement inside the function that declared them 局部有效範圍：由函數中的宣告敘述保留空間

- A variable with **global scope** has storage created for it by a declaration statement located outside any function 全域有效範圍：由不在任何函數內的宣告敘述保留空間
Variable Scope (continued)

Figure 6.15  The three storage areas reserved by Program 6.15
Variable Scope (continued)

Figure 6.16 Relating the scope and type of a variable

C++ for Engineers and Scientists, Third Edition
Program 6.15

```c++
#include <iostream>
using namespace std;

int firstname; // create a global variable named firstname

void valfun(); // function prototype (declaration)

int main()
{
  int secnum; // create a local variable named secnum
  firstname = 10; // store a value into the global variable
  secnum = 20; // store a value into the local variable
  cout << "From main(): firstname = " << firstname << endl;
  cout << "From main(): secnum = " << secnum << endl;
  valfun(); // call the function valfun
  cout << "\nFrom main() again: firstname = " << firstname << endl;
  cout << "From main() again: secnum = " << secnum << endl;
  return 0;
}

void valfun() // no values are passed to this function
{
  int secnum; // create a second local variable named secnum
  secnum = 30; // this only affects this local variable's value
  cout << "\nFrom valfun(): firstname = " << firstname << endl;
  cout << "From valfun(): secnum = " << secnum << endl;
  firstname = 40; // this changes firstname for both functions
  return;
}
```
Scope Resolution Operator

- When a local variable has the same name as a global variable, all references to the variable name made within the local variable's scope refer to the local variable.

如果變數名稱相同先用區域變數

Program 6.16

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

double number = 42.8;      // a global variable named number

int main()
{
    double number = 26.4;  // a local variable named number
    
    cout << "The value of number is " << number << endl;
    
    return 0;
}
```
Scope Resolution Operator (continued)

- To reference a global variable when a local variable of the same name is in scope, use C++’s `scope resolution operator`, which is :: 強制使用全域變數的符號

Program 6.16a

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

double number = 42.5; // a global variable named number

int main()
{
    double number = 26.4; // a local variable named number

    cout << 'The value of number is "' << ::number << endl;

    return 0;
}
```
Misuse of Globals 全域變數的誤用

- Global variables allow programmers to “jump around” the normal safeguards provided by functions 全域變數使得程式設計師避開函數的保護
- Instead of passing variables to a function, it is possible to make all variables global: do not do this 不要把所有變數改為全域變數
  - Indiscriminate use of global variables destroys the safeguards C++ provides to make functions independent and insulated from each other 全域變數的濫用會破壞函數的安全屏障
  - Using only global variables can be especially disastrous in large programs with many user-created functions

C++ for Engineers and Scientists, Third Edition
Variable Storage Categories

• A variable’s scope can be thought of as the space in the program where the variable is valid. 變數有效範圍可以想成變數可用的空間

• In addition to space dimension represented by scope, variables have a time dimension that refers to the length of time storage locations are reserved for a variable. 空間向度和時間向度

• This time, dimension is referred to as the variable’s lifetime. 變數的生命期

• When and how long a variable’s storage locations are kept before they are released can be determined by the variable’s storage category. 儲存類別說明何時和多久一個變數儲存區域可以被保留
Variable Storage Categories (continued)

• The four available storage categories are:
  - auto
  - static
  - extern
  - register

```c++
auto int num;       // auto storage category and int data type
static int miles;  // static storage category and int data type
register int dist; // register storage category and int data type
extern int volts; // extern storage category and int data type
auto float coupon; // auto storage category and float data type
static double yrs; // static storage category and double data type
extern float yld; // extern storage category and float data type
auto char inKey;   // auto storage category and char variable type
```
Local Variable Storage Categories

- Local variables can be members only of the auto, static, or register storage categories
- auto is short for automatic

- Storage for automatic local variables is reserved or created automatically each time a function
  - As long as the function hasn’t returned control to its calling function, all automatic variables local to the function are “alive”

- A local static variable isn’t created and destroyed each time the function declaring it is called
  - Local static variables remain in existence for the program’s lifetime
Local Variable Storage Categories
(continued)

Program 6.17

#include <iostream>
using namespace std;

void testauto();  // function prototype

int main()
{
    int count;  // count is a local auto variable

    for(count = 1; count <= 3; count++)
    {
        testauto();
    }

    return 0;
}

void testauto()
{
    int num = 0;  // num is a local auto variable
    // initialized to 0
    cout << "The value of the automatic variable num is "
    << num << endl;
    num++;

    return;
}
Local Variable Storage Categories (continued)

Program 6.18

#include <iostream>
using namespace std;

void teststat(); // function prototype int main()
{
    int count; // count is a local auto variable

    for(count = 1; count <= 3; count++)
        teststat();

    return 0;
}

void teststat()
{
    static int num = 0; // num is a local static variable
    cout << "The value of the static variable num is now "
        << num << endl;
    num++;

    return;
}
Local Variable Storage Categories (continued)

- Most computers have a few high-speed storage areas, called **registers**, located in the CPU that can also be used for variable storage. 因為 registers 在 CPU中，所以它們比正常記憶體快
  - Because registers are located in the CPU, they can be accessed faster than normal memory storage areas located in the computer’s memory unit.
Global Variable Storage

Categories

- Global variables are created by definition statements external to a function.
- By their nature, global variables do not come and go with the calling of a function.
- After a global variable is created, it exists until the program in which it’s declared has finished executing.
- Global variables can be declared with the `static` or `extern` storage category, but not both.
Figure 6.17  A program can extend beyond one file 一個程式可以超出一個檔案

C++ for Engineers and Scientists, Third Edition
Figure 6.18 Extending the scope of global variables

C++ for Engineers and Scientists, Third Edition
Common Programming Errors

- Passing incorrect data types 傳送錯誤資料型態
- Errors that occur when the same variable is declared locally in both the calling and the called functions 同樣變數宣告兩次
- Omitting the called function’s prototype before or within the calling function 忘記函數原型
- Terminating a function header with a semicolon 函數標題用分號結尾
- Forgetting to include the data type of a function’s parameters in the function header 忘記在函數標頭中加入資料型態
Chapter Summary

- A **function** is called by giving its **name** and passing any **data** to it in the parentheses following the name.
- A function’s **return type** is the data type of the value the function returns.
- Arguments passed to a function when it is called must conform to the parameters specified by the function header in terms of order, number of arguments, and specified data type.
Chapter Summary (continued)

- Functions can be declared to all calling functions by means of a function prototype.
- Every variable has a storage category, which determines how long the value in the variable is retained.