第6章

函数

函数的使用是模块化程序写作(modularized programming) 的重要方式．以硬件为例，我们已熟知一部汽车可以分为引擎、传动、刹车、悬吊、ABS、空调和转向等模块(module)，各模块各司其职，互相协调，而能共同组成一部安全舒适的交通工具．采用相同的想法，我们也可以把许多重要的数据处理程序区分出来，分别包装成函数(function) 的形式，独立进行开发和测试，再以调用的方式使用，降低程序开发的难度．

6.1 函数的基本概念

使用函数的目的

把许多相关的程序指令放在一起，并加以命名，形成一个执行单元，是个非常有用的程序写作方式。 C++ 称呼这种执行单元为 **function** (函数)。其它的语言，例如 FORTRAN 和 BASIC 则分别称之为 subroutine 和 subprogram (子程序)。使用 function 可使程序区分出许多功能明确的组成部分，简化程序写作，减少错误。

事实上，我们在第 2 章里已初步使用过 `fabs(x)` 和 `sin(x)` 等数学运算上不可或缺的函数，而主程序 `main(x)` 本身便是一个标准的函数。函数使用有两大目的：

1. 减少重复撰写类似功能的程序——将常用的算法包装成函数，以利重复使用，简化程序的撰写。
2. 易于调试和维护——把程序的各主要部分区分成几个模块，可以分别进行开发和测试。

函数的语法

函数间可以互相调用。发起调用动作的函数称为调用函数（calling function），被调用的函数称为被调用函数 (called function)，以中文称呼时较不易区别。

以程序开发的观点来看，函数的语法可以分为下列三个部分：

1. 函数的声明
   建立一个函数的原型（prototype），以告知编译器本程序即将使用的函数名称，以及进出这个函数的数据之类型和数量。

2. 函数的定义
   将函数的内容具体地写成程序。

3. 函数的调用
   使用已经定义过的函数。

范例程序

下列程序 TempConv.cpp 是一个使用自定函数的简单程序，包括函数 C2F()的声明、定义及使用．这个程序改写自 5.5 节的程序 Temp.cpp，其功能为产生一个摄氏与华氏温度的对照表:

```
// TempConv.cpp

#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;

// --- 函数 C2F()的声明 ------------------------
float C2F(float);

//------ 主程序 -------------------------------
int main()
 {
   float CTemp;
   cout << "  摄氏    华氏 " << endl ;
   cout << "-----------------"  << endl ;
   for ( int i = 1 ; i <= 10 ; i++ )
   {
     CTemp = 10.0*i;
     cout << setw(5) << CTemp << "  "
          << setw(5) << C2F(CTemp) << endl ;
   }
   cout << "-----------------"  << endl ;
   return 0;
 }

// -------- 函数 C2F()的定义 -------------------
float C2F(float C)
 {
   float F;
```

```
    F = C*9.0/5.0 + 32.0;
    return F;
}
```

操作结果

```
  摄氏    华氏
----------------
   10      50
   20      68
   30      86
   40     104
   50     122
   60     140
   70     158
   80     176
   90     194
  100     212
----------------
```

从程序 Tempconv.cpp，我们可以清楚区分函数 C2F 的声明、定义和使用。

1. 函数 C2F() 的声明

函数在调用使用前要先声明，这点和变量在使用前需声明的情形非常类似．函数的声明(declaration)又称为函数的原型(prototype)，它声明了函数在调用时应该给予的数据类型（如果数据的数目不止一个，则各数据的给予次序也必须一致)，以及执行后所返回的数据类型和次序．例如，Tempconv.cpp 中的函数原型

```
float C2F(float);
```

声明了函数 C2F() 期望收到一个 float 数值，且将返回一个 float 数值，具体表示如图 6.1.1 所示：

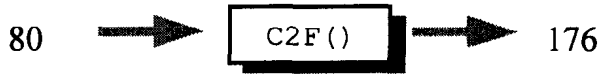

图 6.1.1　函数使用示意图

函数原型的通式为：

返回数据类型 函数名称（参数数据类型列表）;

其中的小括号()称为函数调用运算符（function call operator）。函数原型的放置位置通常在主程序 main( ) 之前，可供主程序和同文件的其它函数调用，如果放在 main( )的主体大括号 {} 内，则只能从 main( ) 内调用。

以下列出几种常见的函数原型：

```
int Wanring( );
int MaxInt(int, int, int);
int Area(int width, int Length);
Area_2(int, int);
void Swap(double x, double y);
```

从上述五例中，我们可以发现参数数据类型列表 (list of parameter data types) 中可以没有任何符号。例如

```
int Wanring( );
```

代表不需要在调用时给予参数。但是它也可以有一个以上的参数。例如

```
int MaxInt(int, int, int);
```

在参数数据类型列表中的各个数据类型后面也可以加入参数名称以帮助这个函数的使用者了解。以本例而言，函数 C2F() 的原型可以写成

```
float C2F(float C);
```

返回数据类型只能有一个。如果执行后不返回任何数据，则以 void 标示。

例如

```
void Swap(double x, double y);
```

此外，int 为缺省的返回数据类型，因此如果返回的数据类型为 int，就可以忽略不写．例如

```
Area_2(int, int);
```

提示

由于主程序 main() 本身亦为函数，因此可以忽略 main 前面的 int 字样．

2. 函数 C2F() 的定义

将函数处理数据的细节写成程序称之为函数的定义 (definition of a function)． 函数一旦完成定义就可以随时调用．函数定义的语法和函数的声明一样，只是把原先函数调用运算符 () 后面的空语句 ";"（null statement）换成具有完整内容的复合语句（compound statement），而且把只有数据类型的参数数据类型列表加入完整的参数名称成为参数列表 (parameter list)．此外，每一个函数，包括 main( )，地位都是相等的，因此，不可以把函数的定义置于另一个函数的定义内．

函数定义的一般语法如下所示：

```
返回数据类型  函数名称（参数列表）
        {
           主体语句
           return 传回值;
        }
```

"返回数据类型　函数名称（参数列表）"的部分称为函数的标头部分（header line），而"{主体语句 return 传回值;}"的部分称为函数的本体（function body）．

例如，程序 TempConv.cpp 中的 C2F( ) 函数定义写成：

```
float C2F(float C)                    // 标头部分
{                                     ┐
  float F;                            │
  F = C*9.0/5.0 + 32.0;               ├  函数的本体
  return F;                           │
}                                     ┘
```

这个函数非常简单，甚至可以不用定义变量 F，而直接将函数定义进一步简化为

```
float C2F(float C)
{ return C*9.0/5.0 + 32.0; }
```

提示

由于 2 的英文发音与 to 相同，因此函数名称使用上述方式命名，以简化程序名称．这里 C2F 很自然地表示 C 和 F 之间的转换 (代表 Centigrade to Fahrenheit)．

3. 函数 C2F() 的调用和使用

调用函数时，必须给予合乎参数列表规定的数据，称为参数（arguments）． 从返回数值的有无，函数的调用可以分为下列两类：

(1) 没有返回数据时，只用函数名称及函数调用运算符 () 以及调用运算符内的参数，就可自成一个完整的语句．例如：

```
swap(a, b);
```

(2) 如果有返回数据，则可以将函数的调用放在任何适合该数据类型的语句中，例如赋

值语句（assignment statements）或算术语句 (mathematical statements):

```
N = MaxInt(P, q, r);
M = 2.5 * Area(W,L);
cout << "Temperature is" << C2F(Temp)
     << " Fahrenheit"    << endl;
```

讨论　实参(parameter)和参数(arqument)

在函数的声明和定义中，参数列表中所声明的变量称为参数。一般而言，参数是函数本体内的局部变量。

当使用函数时，在函数的调用语句中所传递的值称为实参。实参可以是常数，也可以是已经确定的变量或语句。在进行函数的调用时，各参数分别获得各实参的值。例如，在上述的例子中：

swap(a,b)

此调用语句中的 a 和 b 都是实参，而

float C2F (float C) {...}

函数定义式中的 C 是参数。

传值调用

当函数被调用时，参数首先被复制到内存的一个特殊区域，称为堆栈（stack）（下图中的步骤①），且程序的执行次序和控制权都转交给被调用的函数。如图 6.1.2 所示（在这里 `main()` 为调用函数，`C2F()` 为被调用函数）：

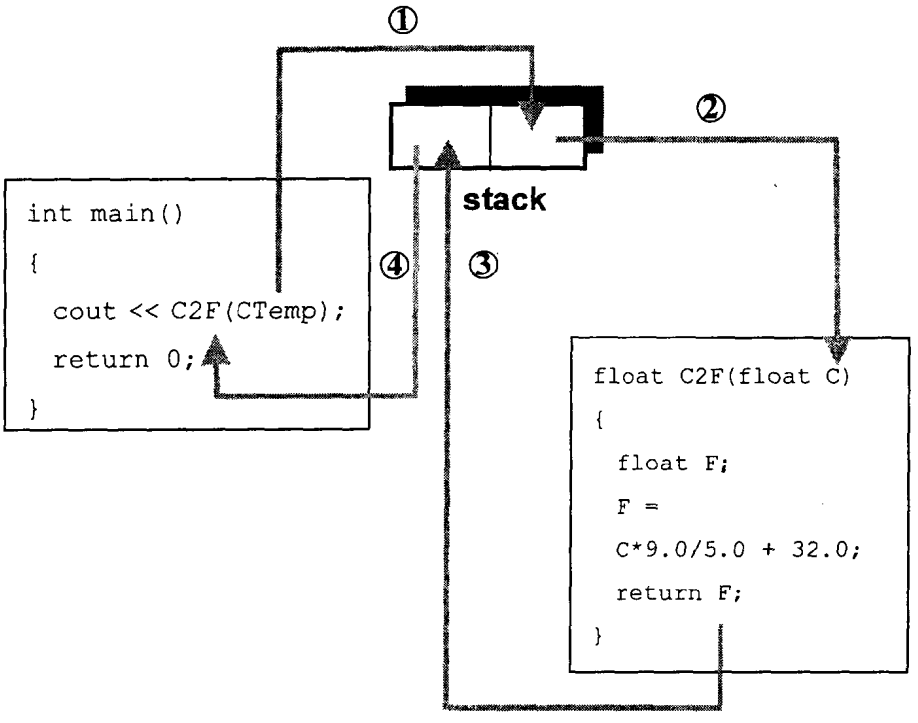

图 6.1.2　函数参数的传递方式

这种参数传递的方式叫做传值 (pass by value) 或以值调用 (call by value)。被调用的函数先将参数 (float C) 初始化为堆栈内的值 (上图中的步骤②)，才开始进行运算。在运算完毕后 (执行完 return F; 或已经到达被调用的函数结尾的地方)，才将结果复制一份到堆栈内 (上图中的步骤③)，接着将控制权交还给调用函数。调用函数这时再把堆栈内的结果复制一份回来 (上图中的步骤④)，并恢复原先的执行次序。在本例中调用函数为 main()，但也可以是别的函数，也就是说函数可以调用函数，执行的步骤和上述程序相同。

由于这种传值的调用方式，参数的值被层层复制，因此被调用函数内对参数的任何更动都不会影响调用函数内的参数。例如，把程序改写成

```
float C2F(float C)
 {
   float F;
   F = C*9.0/5.0 + 32.0;
   C = 3.0;    // 更动 C 的值
   return F;
 }
```

则主程序内的 CTemp 并不会跟着改变。也就是说，通过传值的机制，所有函数的输出和输入都在严格的控制之下，不会造成意料之外的更动。

不使用函数原型 (prototype) 的语法

函数原型 (prototype) 的目的在于声明 (declare) 一个 (在程序的先后次序上) 尚未定义但即将使用的函数。如果我们把函数定义置于函数原型的位置 (亦即在 main() 之前)，则此函数定义就兼具声明和定义的功能，不需要再使用函数原型了。

例如以下的程序 TempConv2.cpp 和原先程序 TempConv.cpp 在功能上没有任何差异。

范例程序 文件 TempConv2.cpp

```
// TempConv2.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;

// 以下为函数 C2F()的定义，兼具声明的功能
//-------------------------------------
float C2F(float C)
 {
   float F;
   F = C*9.0/5.0 + 32.0;
   return F;
```

```
}
// 以下为主程序
//-------------------------------------
int main()
{
  float CTemp;
  cout << "  摄氏   华氏 " << endl ;
  cout << "-----------------"  << endl ;
  for ( int i = 1 ; i <= 10 ; i++ )
  {
    CTemp = 10.0*i;
    cout << setw(5) << CTemp << "  "
         << setw(5) << C2F(CTemp) << endl ;
  }
  cout << "-----------------"  << endl ;
  return 0;
}
```

函数内 return 的功能

我们在这里通过另一个范例程序讨论函数内“return 语句”(return statement) 的功能。

范例程序　文件 SeasonsFnc.cpp

下列完整程序 SeasonsFnc.cpp 改写自第 4 章 Seasons.cpp。这个程序把计算季节的部分写成函数 CheckSeason():

```
// SeasonsFnc.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

// 以下为函数 CheckSeason()的声明
void CheckSeason(int);
```

```
//------ 主程序 ---------------------------------
int main()
{
  int M;
  cout << "\n"
       << "请输入一个月份  :  " << endl;
  cin  >> M;
  CheckSeason(M);
  return 0;
}

// --- 以下为函数 CheckSeason()的定义 ------------
void CheckSeason(int Month)
{
  if (Month < 1 || Month >12)
    { cout <<  "您输入的月份没有意义!";    return;   }
  cout << "\n" << Month << "月是";
  switch ((Month%12)/3)
  {
    case 0:
      cout << "冬季" << endl;
      break;
    case 1:
      cout << "春季" << endl;
      break;
    case 2:
      cout << "夏季" << endl;
      break;
    case 3:
      cout << "秋季" << endl;
      break;
    default:
      cout << "程序有问题!"  << endl;
   }
return;
}
```

讨论

函数 CheckSeason() 中使用了两个 "return;"。第一个 return 放在 if 语句中，一旦成立，(Month < 1) 或 (Month >12)，便返回调用函数 main()。这种写法避免了第 4 章 Seasons.cpp 中使用 if-else 语句将 switch 语句包起来的做法，也不需要如 SeasonsGoTo.cpp 中使用 goto 语句的写法，但可以获得完全一样的结果。

6.2 以引用的方式调用 (call by reference)

使用上一节的基本方式，我们已经可以写出很多实用的程序。但是，我们也注意到基本的函数受到只能传回一个数值的限制。此外，有些时候我们也想改变参数的数值。要突破这些限制，可以使用引用（reference）的技巧。

引用是同一个变量的别名（alias）。例如，下列两个语句：

```
int N;
int& M = N;
```

定义了变量 N，以及 N 的引用 M。此外这里 & 称为引用运算符（reference operator），表示 M 和 N 所代表的变量位于同一个地址上。对于 M 所作的变化都等同于直接作用在 N 上，反之亦然。"int& M = N;" 这个定义引用的语句也可以写成

```
int &M = N;
```

也就是说，下面两个函数的标头部分（header line）是一样的：

```
int Fnc(int& N; float& x)
int Fnc(int &N; float &x)
```

提示

上面介绍的引用运算符和取址运算符 (address operator，我们将在第 8 章中介绍) 都使用相同的符号 “&”，但是他们的用法和意义却不一样．例如：

```
int N;
int M = &N;
```

所代表的意义是：“把 N 的地址（也是 32 位大小）存放到整数 M 里”．

在下列程序 Alias.cpp 中，我们定义了变量 N，以及 N 的引用 M，以了解它们之间的关系．

范例程序 文件 Alias.cpp

```
// Alias.cpp
#include <iostream>
using namespace std;
// ------ 主程序 ------------------------
int main()
 {
    int N = 10;
    int& M = N;
    cout << "M 的值原来是：" << M << endl;
    cout << "N 的值原来是：" << N << endl;
    N = 5;
    Cout << "执行 “N = 5;” 之后" << endl;
    Cout << "M 的值目前是：" << M << endl;
    M = 2;
    Cout << "执行 “M = 2;” 之后" << endl;
    Cout << "N 的值目前是：" << N << endl;
    return 0;
 }
```

操作结果

```
M 的值原来是：10
N 的值原来是：10
执行 “N = 5;” 之后
M 的值目前是：5
执行 “M = 2;” 之后
N 的值目前是：2
```

由于引用所代表的是同一个变量，因此，6.1 节程序 TempConv.cpp 中的函数 C2F() 也可以进一步改写，使用引用将数值传回。如下列程序 TempConv3.cpp 所示：

范例程序　文件 TempConv3.cpp

```
// TempConv3.cpp
#include <iomanip>
using std::cin;
using std::cout;
using std::endl;
using std::setw;
void C2F(float, float&); // 函数 C2F() 的原型
//-------- 主程序 ------------------------------
int main()
 {
   float CTemp, FTemp;
   cout << "  摄氏    华氏 " << endl ;
   cout << "-----------------" << endl ;
   for ( int i = 1 ; i <= 10 ; i++ )
   {
     CTemp = 10.0*i;
     C2F(CTemp, FTemp);
     cout << setw(5) << CTemp << "  "
```

```
            << setw(5) << FTemp << endl ;
    }
    cout << "-----------------"  << endl ;
    return 0;
 }

//-------- 函数C2F()的定义 ---------------------
void C2F(float C, float& F)
 {
   F = C*9.0/5.0 + 32.0;
   return;
 }
```

操作结果

```
  摄氏   华氏
-----------------
   10     50
   20     68
   30     86
   40    104
   50    122
   60    140
   70    158
   80    176
   90    194
  100    212
-----------------
```

讨论

1.在程序 TempConv3.cpp 中，函数 C2F( ) 中的第二个参数 F 为主程序中变量 FTemp 的引用 (reference)，因此不需要回传任何数值，只要在函数 C2F( )中改变 F 的值，就可以直接反映在主程序内的变量 FTemp 上。

2.函数 C2F()定义中的“return;”可以省略。程序执行到被调用函数结尾的地方，会自动返回调用函数。

一般而言，为了避免在不经意的情况下更动到调用函数内的变量，我们并不鼓励这样的做法，而将引用的使用限制在下面三种情况：

1. 参数本身必须改变。
2. 要传回两个以上的值。
3. 有大量参数需要传递。如果不使用引用，则将耗费时间在参数的复制上。例如，参数为大型向量或矩阵的情况。

程序 Swap.cpp 是一个典型的程序，利用引用来交换两个变量的值。

范例程序　文件 Swap.cpp

```
// Swap.cpp
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

void Swap (int&, int&);
void SWAP2 (int, int);
//----------------------------------------
int main()
 {
   int A = 5, B = 10;
   Swap(A, B);
   cout << "执行过 Swap()\n";
```