第 5 章 函数

【学习内容】

本章将介绍 C++ 的函数机制，学习模块化程序设计思想。主要内容包括：

- 函数的定义和函数原型。
- 函数调用和参数传递机制。
- 函数重载。
- 存储类别和作用域。
- 递归函数设计和函数的递归调用。
- 预处理指令。

【学习目标】

- 掌握函数的定义和使用方法，理解函数原型、函数重载。
- 掌握传值和传引用两种参数传递的机制。
- 理解存储类别和作用域。
- 理解函数递归调用的执行过程。
- 能够熟练利用函数进行自顶向下、逐步求精的程序设计。

通过前面的学习，我们已经可以设计程序了，但能设计的程序都非常简单，真正有实际应用价值的程序都比较复杂。那么，如何设计复杂的程序？如何组织大型的软件呢？通过本章的学习，读者将了解函数机制，掌握通过函数实现程序的模块化的方法。

5.1 模块化程序设计

前面介绍过模块化的思想：对于一个复杂问题，将其分解成若干稍简单的问题；解决这个复杂问题时，要把它分解成若干稍小的、简单的部分，每部分对应一个模块进行问题求解。因此，要设计一个解决复杂问题的较大的系统，可以按子结构之间联系的疏密程度分解成较小的模块，每个模块完成一定子问题的求解，整个程序是由层次的逐级抽象的诸模块组成。在 C++ 程序中，每个模块可以是一个类或函数。

函数是 C 程序构成的基础，C 程序就是由一个个函数组成的。由于 C++ 是 C 的超集，所以一个 C++ 程序也可以像传统的 C 程序一样，也由一个个函数组成。但是，采用面向对象程序设计方法设计的 C++ 程序是由一个个类构成的，即便如此，任何一个 C++ 程序也至少包含一个函数——main 函数。函数也是类的方法的实现手段。因此，在 C++ 中，函数的作用有两

个：一是按功能将一个复杂的大任务划分为若干小任务时，这些小任务可用函数来实现；二是在类中，用函数来定义类对象的方法，详见第 10 章及其后续章节。一个 C++ 程序无论有多么复杂、规模有多么大，程序的设计最终都落实到一个个函数模块的设计上，通过对函数模块的调用实现程序的特定功能。

函数是 C++ 源程序的基本模块，C++ 中的函数相当于其他高级语言中的子程序。C++ 中的函数包括两类：预定义函数和用户自定义函数。C++ 提供了极为丰富的库函数，这些库函数是预定义好的，程序员可以在自己的程序中直接使用。当然，还需要将包含欲使用的函数声明的头文件包括进来。另外，C++ 还允许用户根据需要建立自定义函数，把自己的算法编写成一个个相对独立的函数模块，然后像调用库函数一样来调用这些函数。程序一般都由程序员编写的各种自定义函数与 C++ 标准库中提供的预定义函数组合而成。

C++ 继承了 ANSI C 语言在结构化程序设计上的优点，通过采用函数模块式的结构，C++ 语言易于实现结构化程序设计，使程序的层次结构清晰，便于程序的编写、阅读和调试。其实，结构化程序设计的方法也是面向对象程序设计的基础，因为在面向对象程序设计中，一个类的方法的实现，最后还是要使用结构化程序设计手段。本章重点学习如何用 C++ 的函数机制实现结构化和模块化的程序设计思想。

使用 C++ 进行模块化程序设计时，一个 C++ 完整的程序可包含多个文件，一个文件中可包含多个函数。此外，文件是通常意义下的“模块”，它也是编译单位。因此，当某一文件内容改变时，只需单独编译此文件，然后将它们与其他编译过的文件进行连接，就可生成可执行程序。

5.2 预定义函数的使用

C++ 提供了丰富的预定义函数，也称为库函数。这些库函数的声明（一般称为函数原型）都放在相应的头文件中，库函数的声明说明了库函数的名字、参数个数及类型、函数返回结果的类型等信息。使用这些库函数时，首先要将相应的头文件包括进来，然后就可直接使用库函数了。例如，数学库函数中的函数能完成一般数学运算，其声明包含在 math.h 中，使用这些函数时应加下面的代码。

```
#include <math.h>
```

如第 2 章介绍，这是早期的 C++ 语言为了跟 C 语言相融合，头文件使用带后缀“.h”的形式。但新的 C++ 标准已经不推荐使用，直接删除了后缀“.h”，如 #include <iostream>。而为了兼容 C 语言一些比较重要库函数的头文件，采取的方法是去掉后缀“.h”，并在文件名前面加上前缀 c，表示其中的函数来自 C 标准库。例如，C++ 版本的 math.h 对应的是 cmath，即 C++ 新标准引用数学库的方式是：

```
#include <cmath>
```

在程序中要调用库函数时，遵循的调用格式如下：

```
<函数名> (<参数列表>)
```

上述格式中括号内是提供给被调用函数的参数，称为实在参数（actual arguments），简称

实参。如果有多个参数,参数之间用逗号分隔,提供的参数可以是常量、变量和各种表达式。要求调用函数时提供的参数在数目、类型上必须与函数的声明一致。函数可以有一个返回结果,所以函数调用本身可以构成表达式的一部分。

函数调用的一般过程是:先计算实参表达式的值,将计算的结果交给被调用函数(通过后面介绍的参数传递机制);接着执行被调用函数的代码;直至执行到返回语句 return 或执行到函数末尾,程序控制返回到调用函数的调用处,继续执行。

例如,求平方根函数 sqrt 的声明在 cmath 中,该函数接受一个 double 类型的参数,函数的返回结果也是 double 类型,假设 main 函数中有下面的代码:

```
printf("%.2f", sqrt(900.0));
```

执行过程是:以 double 类型的 900.0 为参数调用 sqrt 函数,该函数计算参数 900.0 的平方根,返回结果 30.0;然后,以%.2f 和 30.0 为参数调用 printf 函数,实现格式化输出。这里 printf 也是预定义库函数,其声明在 cstdio 中。

除了预定义的库函数外,程序员还可根据实际需要、按照模块化原则设计自己的函数,然后通过调用这些函数完成相应的功能。下面介绍用户自定义函数的定义和使用。

5.3 函数定义与函数原型

函数是一个命名的程序代码块,是程序完成其操作的功能单位。在程序设计中,有许多算法是通用的。例如,求一个数的平方根,解一元二次方程。如果将这些算法定义为一个函数,就可以在程序中需要这些算法的地方直接使用它们(而不必再进行定义)。C++ 的库函数已经提供了丰富的功能,但在进行程序设计时,程序员往往还需要根据具体问题的需求设计自己的函数模块。为此,应该先定义一个函数,然后就可以像使用预定义函数一样使用自定义函数。定义一个函数就是编写完成该函数功能的程序块。

5.3.1 函数定义

1. 函数定义格式

函数定义的一般格式如下:

```
<返回值类型> <函数名>(<参数列表>)
<函数体>
```

说明:

- <返回值类型>说明函数返回值的数据类型,简称函数的返回类型。它可以是任一基本数据类型或用户自定义的数据类型;如果无返回值,则用关键字 void 说明。默认的返回类型是 int,即若未指定返回类型,则返回类型是 int(注意不是 void)。
- <函数名>是程序员为该函数指定的名字,函数名应遵守标识符命名的规定。
- <参数列表>指明参数的个数、名称和类型,多个参数之间用逗号分隔。函数定义中的参数称为形式参数(formal parameters),简称形参,因为在函数没有调用时,形参还没有分配对应的存储空间,所以也称为虚拟参数。如果函数没有形参,参数列表为空,函数名后面的圆括号不能少,这时也可在括号中加上关键字 void,表示这是一个无参

函数。

- ＜返回值类型＞、＜函数名＞及＜参数列表＞构成了函数头。
- ＜函数体＞描述函数的功能，即函数所完成的具体操作。它由一系列说明语句和执行语句组成。函数体实际上是一个复合语句，花括号不能少，它指明函数体的开始和结束。函数执行时，如同执行一个复合语句一样，顺序执行函数体，直到遇到 return 语句或者遇到表示函数体结束的那个右花括号为止，函数执行完毕后，返回调用程序继续执行。

例如：

```
int max(int a, int b, int c)
{
    int m;
    m = (a > b) ? a : b;
    return (m > c) ? m : c;
}
```

定义了一个名为 max 的函数，该函数有 3 个形式参数 a、b 和 c，类型均为 int，返回值也是 int 类型。函数体由一个变量说明语句、一个赋值语句和一个返回语句构成。函数的功能是求 a、b 和 c 中最大的值。

兼容性提示： 和 C++ 不同，在 C 语言中，空的参数列表表示函数可以接收任意多个参数，参数列表必须为 void 才表示无参函数，所以对于 C 语言的无参函数最好在参数列表中写上 void。

所有函数的定义是并列的、平行的，不支持函数嵌套，即不允许在一个函数定义内部定义另外一个函数。但是，可以对别的函数进行调用或作引用说明。

函数定义中声明的所有变量都是局部变量(local variable)，局部变量只在声明语句所处的程序块中有效。大多数函数都有一组参数，函数定义时指明了每个形式参数的名字、类型，形式参数与函数调用时提供的实在参数一一对应，函数执行时，形式参数可以视为实在参数的代表。参数机制提供了函数之间沟通信息的方式。函数的形式参数也可视为局部变量，在函数体范围内有效。

2. 函数的返回值

根据返回值的情况，函数可以分为两类：无返回值的函数和有返回值的函数。

(1) 无返回值的函数：函数无返回值时，返回类型必须用关键字 void 说明。这类函数的函数体内无 return 语句或 return 语句中无表达式。

如果函数体中包含不带表达式的 return 语句，当函数体执行到这样的 return 语句时，即返回到原先的调用位置；如果函数体无 return 语句，当执行完函数体最后的语句，遇到表示函数体结束的右花括号时，返回到原调用位置。

(2) 有返回值的函数：函数返回值的类型如果不是 void，表示该函数有返回值，此时函数体必须包含带表达式的 return 语句。表达式的值将返回给调用程序，该值具有函数说明的返回值类型。函数返回值的类型可以是算术类型，也可以是后面将介绍的指针、引用及用户定义的数据类型(如结构、联合或类等)。

不论函数有无返回值，函数体可有多个 return 语句。函数调用时，只要遇到一个 return 语句，就将忽略函数体中其余的代码，立刻返回到调用程序。例如，上面定义的函数 max 计算并返回 3 个整数中的最大值。

5.3.2 函数原型

函数一经定义,就可在程序中多次使用,函数的使用是通过函数调用实现的。而对于编译器来说,则要求在调用某个函数之前先定义该函数,以便编译器对函数调用的合法性进行全面检查。这种要求则对程序代码结构组织带来很多限制,甚至是难以完成的任务。

为解决上述问题,C++ 引入函数原型(function prototype)的概念。跟变量声明类似,函数原型作用是声明函数,目的是告诉编译器该函数的函数名字、函数返回的数据类型、函数要接收的参数个数、参数类型和参数顺序。编译器通过函数原型来验证函数调用的正确性。

函数原型的基本形式如下:

```
<返回值类型>  <函数名>(<参数列表>);
```

函数原型看上去只是比函数头多了一个分号,实际上,函数原型的参数列表中形参的名称可以省略。因为函数原型的意义在于验证函数调用时实在参数和形式参数的一致性,而不用关心形式参数的名字,因而只要指明参数的个数、类型及顺序就足够了。

例如,函数 max 的原型是:

```
int  max(int , int , int);
```

这个原型表示函数 max 具有 3 个 int 参数,返回 int 类型结果。注意这个函数原型与 max 函数定义的首部相似,只是不包括参数名 a、b 和 c,并且在尾部多了一个分号“;”。

编译器在调用某个函数之前,要求是先给出该函数的定义。有了函数原型之后,函数调用前可以先给出该函数的原型,函数的具体定义在后续或其他代码中提供,如例 5-1 所示。

【例 5-1】 用自定义函数计算并返回 3 个整数的最大值,并输出结果。

```
//ex5_1.cpp:用自定义函数计算并返回 3 个整数的最大值,并输出结果
#include <iostream>
using namespace std;

int max(int, int, int);                          //函数原型

int main()
{
    int x, y, z;
    int maximum;

    cout << "Please enter three integers: ";
    cin >> x >> y >> z;

    maximum = max(x, y, z);                      //调用 max 函数
    cout << "The maximum integer is: " << maximum << endl;

    return 0;
}

int max(int a, int b, int c)                     //具体函数定义
{
    int m;
```

```
    m = (a > b) ? a : b;
    return (m > c) ? m : c;
}
```

程序运行结果：

```
Please enter three integers: 12 34 23
The maximum integer is: 34
```

程序设计风格提示：如果有几个函数在同一个文件中，为方便阅读代码，有经验的程序员一般把主函数放在前，而把被调用函数的定义放在后，并将其原型放在主函数前进行声明，以符合“先声明，后使用”的原则。

5.4 函数的调用

5.4.1 函数调用的概念

在C++程序中，除了main函数以外，任何一个函数都不能独立地在一个程序中存在，都是通过main函数直接或间接地调用引发执行的。调用一个函数就是执行该函数的函数体的过程。

函数调用的一般形式如下：

```
<函数名>(<实在参数列表>)
```

<函数名>是用户自定义的函数名或者系统预定义的标准函数名。<实在参数列表>是一个表达式列表。实在参数与形参表中的形式参数要顺序一致、类型匹配、个数相同(可变参数除外)。如果被调用函数是无参函数，它被调用时就没有实参，但圆括号必须有。如果有形式参数，相应的实在参数应该是类型一致的表达式，包括常量、变量或后面介绍的数组元素等。

C++的函数调用可以出现在表达式中，这种情况下要求被调用的函数应该有返回值。对于没有返回值的函数，也可以单独构成函数调用语句(在函数调用后面加上分号即可)。

函数调用时，先计算每个实参表达式的值，并给每个形式参数分配内存单元；接着将实参的值赋给被调用函数的对应的形参；然后再执行被调用函数的函数体。函数体执行时，顺序执行各语句，直到遇到return语句或者遇到表示函数体结束的右花括号为止，被调用的函数执行完毕，返回调用程序继续执行。如果被调用函数有返回值，系统从被调用函数返回到调用程序时，会将return语句中的表达式的值返回给调用程序，调用程序可以继续使用该值进行后续运算；如果被调用函数没有返回值，则被调用的函数执行完毕时，直接返回到调用程序，继续执行。图5-1给出了调用max函数的示意图。

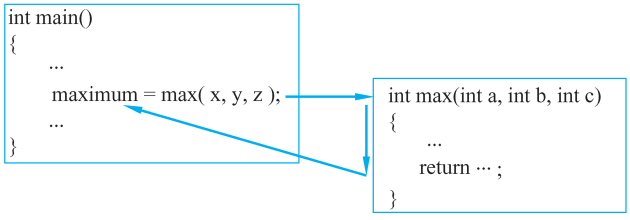

图5-1 调用max函数示意图

如果实在参数与形式参数的类型不同,则需要把实参的类型转换成对应形参的类型。例如,数学库函数 sqrt 可以用整型参数调用,虽然 cmath 中的函数原型指定为 double 参数,但函数仍然可以顺利工作。下列语句

```
cout << sqrt(4);
```

先隐式地将 int 类型的 4 转换成 double 类型的 4.0,然后再用 4.0 调用 sqrt 函数,所以能正确地求出 sqrt(4)的值为 double 类型的 2.0。一般来说,与函数原型中参数类型不完全相符的参数值将被隐式地转换为正确类型之后才能进行函数调用。

将数值转换为较低类型时可能导致数据丢失精度。为了避免在编写调用代码时出现这类错误,C++ 不会自动将精度较高类型的实在参数自动转换成精度较低类型的形式参数。如果确实要将精度较高类型的实在参数值传递给精度较低类型的形式参数,只能先显式地将该值赋给较低类型的变量,再用该变量作为实在参数来调用函数,或者在实参表达式中使用强制类型转换运算符实现类型转换。这种要求可以看成保证程序安全性的一种措施。

例如,对于例 5-1 的 max 函数,3 个形式参数 a、b 和 c 都是 int 类型,如果变量 x、y 和 z 是 double 类型的,下面对 max 的调用方式 1 和调用方式 2 是正确的,而调用方式 3 将导致警告性错误。

调用方式 1:

```
max((int)x, (int)y , (int)z);
```

调用方式 2:

```
int nx=(int)x;
int ny=(int)y;
int nz=(int)z;
max(nx, ny, nz);
```

调用方式 3:

```
max(x, y, z);
```

【例 5-2】 用自定义函数求两个整数的最大公因子,并编写程序调用该函数输出结果。

设计思想:本题解法可分为 3 步:①输入数据;②求最大公因子;③输出结果。核心问题是如何设计一个函数求两个整数的最大公因子。求最大公因子有多种方法,下面的辗转相除法是其中一种。

对于 a 和 b 两个整数,假设 a>b,如果 a 不能被 b 整除(余数记为 remainder),则反复执行下面的步骤,直到 a 能被 b 整除。

```
a = b;
b = remainder;
remainder = a % b;
```

运算结束后 b 的值就是两个输入整数的最大公因子。程序如下实现。

```
//ex5_2.cpp:用自定义函数求两个整数的最大公因子
#include <iostream>
```

```
using namespace std;

int gcd(int a, int b);                          //gcd 函数的原型

int main()
{
    int x, y;
    int fac;

    cout << "Please enter two integers: ";
    cin >> x >> y;
    fac = gcd(x, y);                            //调用 gcd 函数
    cout << "The greatest common divisor of ";
    cout << x << " and " << y << " is: " << fac << endl;

    return 0;
}

int gcd(int a, int b)                           //gcd 函数的定义
{
    //如有必要,通过交换确保 a>=b
    if (a < b)
    {
        int temp = a;
        a = b;
        b = temp;
    }

    int remainder = a % b;
    while (remainder != 0)
    {
        a = b;
        b = remainder;
        remainder = a % b;
    }

    return b;
}
```

程序运行结果：

```
Please enter two integers: 120 64
The greatest common divisor of 120 and 64 is: 8
```

5.4.2 参数传递

在调用带参数的函数时,需要进行参数传递。参数传递提供了调用函数和被调用函数交换信息的渠道。C++ 中的参数传递默认采用的是传值(call by value)方式,即传入的只是参数的值的副本,这个过程可以理解为,在参数传递时执行了“形式参数＝实在参数”的赋值操作,所以传递之后,形参中存放了实参值的副本,但形参和实参是两个独立的变量。

在被调用函数执行时,只能访问形式参数对应的内存单元,而不能直接访问实在参数对应的内存单元,因而无法直接改变实在参数的值。下面的 swap 函数试图交换两个参数的值。

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

void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}

int main()
{
    int x = 10, y = 20;
    swap(x, y);
    cout << "x:" << x << "y:" << y << endl;
    return 0;
}
```

程序运行结果:

```
x:10   y:20
```

运行结果显示,x 与 y 的值并没被交换,这是因为 main 函数在执行 swap(x, y)时,是把实在参数 x 和 y 的值传给了 swap 中形式参数 x 和 y(注意,实参 x、y 与形参 x、y 虽然名字相同,但其实是不同的变量,在计算机内存中有各自不同的存储单元),swap 函数中交换的也只是形参 x 和 y 这两个局部变量的值,修改的是形式单元中的值,当 swap 函数执行完后,控制返回 main 函数的调用语句处,继续向下执行,此时形参 x 和 y 这两个局部变量也被撤销。可见,在整个 swap 函数执行过程中,形参 x 和 y 数值的改变丝毫不影响调用函数中实参 x 和 y 的值。

从上面的程序可以看出,参数传递允许信息从调用函数流向被调用函数,而被调用函数想要将数据传给调用函数,只能通过返回结果这一途径。在许多情况下,这一限制为程序设计带来了不便,因为它使得从被调用函数向调用函数传递的信息非常有限。为了弥补这一不足,使被调用的函数能够访问调用它的函数的变量,C++ 提供了传引用(call by reference)这一传递参数的方式。

如果采用传引用方式进行参数传递,要求实在参数只能是变量(具有左值性质,即具有对应的内存地址)。当调用一个函数时,是把对实参变量的引用传给形式参数,此时形参与对应的实参可视为是同一变量,或者说形参是实参在被调用函数中的别名。实际上是将实参变量的内存地址存放到对应的形参的内存单元中,当程序在执行被调用函数过程中,对形式参数的任何操作都被处理成对相应实参变量的间接访问。当被调用函数执行完毕返回时,形式参数所指的实参变量就存储了所期望的值。可见传引用的参数传递方式可以通过形式参数访问调用函数中的实参变量,实现信息从被调用函数传向调用函数,从而可以间接实现返回多个值。

为了区分参数不同的传递方式,C++ 规定,对于按照传引用方式传递的参数,对应形式参数名前要加上符号“&”,如果没有该符号,说明该参数按照默认传值方式进行传递。同一函数中,允许有的参数传引用,有的参数传值。

例如,为了实现两个整数的交换,上面的 swap 函数应该采取传引用方式。

【例 5-3】 用自定义函数实现两个整数的交换。

```
//ex5_3.cpp:用自定义函数实现两个整数的交换
#include <iostream>
using namespace std;

void swap(int &, int &);

int main()
{
    int x = 10, y = 20;
    swap(x, y);
    cout << "x:" << x << "   y:" << y << endl;

    return 0;
}

void swap(int &a, int &b)
{
    int temp;
    temp = a;
    a = b;
    b = temp;
}
```

程序运行结果：

```
x:20   y:10
```

可见，采取传引用方式进行参数传递，形参、实参结合时的语义与传值方式的调用是不同的。在这个例子中，形式参数 a 和 b 视为对应的实参变量 x 和 y 的别名，即虽然名字不同，但可以简单认为是同一变量。对形式参数 a 和 b 的任何访问（包括赋值），实际上是对实参变量 x 和 y 的访问，因而可以实现在调用函数过程中对实参变量进行修改。通过这种方式，可以达到间接返回多个值的效果。

与传引用方式类似，C++ 还可以通过传递指针参数来实现被调用函数访问调用它的函数的变量。这部分内容将在第 7 章中详细介绍。

对比传值和传引用：传值和传引用这两种方法各有优缺点。使用传值方式，可通过函数的返回值显式地将数据交给调用程序，从而改变调用程序中变量的值，但它不能返回多个值，返回值的类型也受到限制。采取传引用方式，可将更多的数据从被调用函数传向调用函数，但它不是显式地修改调用程序中的变量，损害模块的独立性，易导致一些潜在的错误。

5.4.3 默认参数

为了方便程序设计，在函数调用时可能要为某些参数传递特定的值。程序员可以将这些参数指定为默认参数，并提供这些参数的默认值。当函数调用时，如果没有为默认参数提供相应的实参，系统将把预设的默认参数值传递给被调用函数的形参。

默认参数应在函数名第一次出现时（如函数原型或具体函数定义）指定，通常是在函数原型中，直接在参数说明中增加一个赋值符号和默认值，默认值可以是常量、全局变量或函数调用。