第5章 函 数

一般来说，要解决的问题或任务很复杂，通常把它们分解成多个小问题或子任务，并为每个小问题或子任务定义一个独立的函数。每一个函数都有清楚的核心任务并且都不太长。在C程序中，一个程序可以由多个函数构成，每个函数都有独立的功能。

本章将介绍函数的概念、定义及函数的调用方式。

5.1 初识函数

本书的前面各章节，编制的程序仅包含一个main函数，所有的代码都写在main函数体内。如果我们要解决的问题或任务很复杂，main函数内的代码就会很多，无论是写程序的人还是看程序的人，都会感到阅读困难。写程序与写文章有相似之处，如果一篇文章从头到尾只有一个段落，虽然同样可以表达文章的中心思想，但是又有多少读者愿意耐心的看完如此庞大的一个段落呢？因此，编制程序时，如果将一个程序适当的划分为若干函数，可以提高程序的可读性。

5.1.1 函数的分类

函数的概念虽然是初次出现，但是读者从本书第一个例子起就开始使用函数了。下面先来再看看本书的第一个C程序。

【例5.1】 原样输出一行语句。

```
#include<stdio.h>                          /*输入输出函数编译预处理命令*/
void main()                                /*主函数*/
{
    printf("Hello,world!\n");              /*输出信息*/
}
```

程序运行结果如下：

```
Hello,world!
```

分析与说明：

(1) 该程序仅包含一个 main 函数,main 函数是由系统定义的。

(2) main 函数体内的唯一一条语句是调用 printf 函数,它是由系统提供的标准格式控制输出函数。

printf 函数是由系统提供的库函数,用户可以直接调用,而在实际编程中,为了解决问题,需要自己定义函数来实现所需的功能。接下来,看一个包含自定义函数的例子。

【例 5.2】 一个包含自定义函数的例子。

```
/*头文件包含的预处理命令*/
#include<stdio.h>
#include<conio.h>
/*用户自定义函数的声明语句。关于函数声明语句的用法及作用将在 5.3.2 节作详细说明*/
void menu();
void welcome();
/*程序的入口处:主函数。每个 C 程序都有且必须有一个主函数。*/
void main()
{
    menu();                              /*调用自定义函数 Menu 显示界面*/
    welcome();                           /*调用自定义函数 welcome 显示欢迎词*/
    menu();                              /*再次调用自定义函数 Menu 显示界面*/
    getch();                             /*调用库函数 getch 接受来自键盘的输入*/
}
/*定义函数实现界面的显示*/
void menu()
{
    /*三次调用库函数 printf 实现界面图案的输出*/
    printf("*****************************************************\n");
    printf("            欢迎进入函数章节的学习                   \n");
    printf("*****************************************************\n");
}
/*定义函数实现欢迎词的显示*/
void welcome()
{
    printf("    welcome!\n");            /*调用库函数 printf 实现欢迎词的输出*/
}
```

程序运行结果如下:

```
*****************************************************
            欢迎进入函数章节的学习
*****************************************************
welcome!
*****************************************************
            欢迎进入函数章节的学习
*****************************************************
```

说明：

(1) 例 5.2 由三个函数组成：主函数 main()、menu()和 welcome 函数()。在程序设计中，为了增加程序的可读性，提高程序开发效率，程序员常将一些常用的功能模块编写成独立的函数，要善于利用函数，以减少代码的重复编写。本例 main 函数中，两次调用 menu 函数正是提高了 menu 函数的利用率。

(2) C 程序的执行总是从 main 函数开始，当执行到函数调用语句时，转入被调函数体内，当被调函数执行完毕，返回到调用点接着执行。

(3) 组成一个 C 程序的所有函数都是互相独立的，函数之间不存在从属关系，在一个函数体内不能定义另一个函数，即函数不能嵌套定义，但可以嵌套调用。

(4) 一个 C 程序可以由一个 main 函数和多个自定义函数构成。main 函数可以调用其他函数，其他函数之间也可以互相调用；main 函数是系统定义的，自定义函数不能调用 main 函数；同一个函数可以被一个或多个函数调用任意多次。

(5) 从用户的使用角度，函数分为以下两类。

① 库函数(标准函数)：如 printf 函数、scanf 函数等由系统提供无须用户定义即可直接调用的函数。

② 用户自定义函数：由用户自己定义实现的函数，如例 5.2 中的 menu 和 welcome 就是用户自定义函数。

(6) 从函数的定义形式，函数分为以下两类(本段文字中涉及的一些概念，例如返回值、实参、形参等，其具体内容会陆续在本章后面的小节中详细介绍，读者此时只需有一个初始印象即可)。

① 无参函数：一般用来执行一组特定的操作，执行后的结果直接在该无参函数中输出，不返回给主调函数，一般没有返回值。主调函数在调用无参函数时，不发生函数间的数据传递。如例 5.2 中的 menu 和 welcome 就是无参函数。

② 有参函数：主调函数调用有参函数时，通过实参和形参进行数据传递，一般都有返回值。

5.1.2 函数的定义

函数定义的一般格式：

```
<类型标识符><函数名>([形式参数列表])        /* 函数首部 */
{                                           /* 函数体 */
    变量说明
    执行语句
}
```

说明：

(1) ＜函数类型＞：每个变量都有对应的数据类型，每个函数也都有对应的数据类型(如 int、float、char 等)。实际上函数的数据类型和该函数的返回值的数据类型是一致的，如果函数没有返回值，函数类型用 void 表示。

(2) 函数的返回值：有的函数有返回值，有的函数没有返回值。函数的返回值是指函数被调用之后，执行函数体中的程序段所取得的并返回给主调函数的值。如调用正弦函数取得正弦值，就是正弦函数的返回值。有返回值的函数，其函数体内必须有相应的返回语句 return，具体语法见下面的例题。

(3) ＜函数名＞：唯一标识一个函数的名称，是一个合法的标识符。

(4) ([形式参数列表])：由 0 个、1 个或多个参数(很多初学者对参数的概念难以理解，实际上参数就是变量，只不过是处于特定位置上具有特定作用的变量而已)组成，多个参数之间用逗号隔开。每个形参对应一个类型说明符。

(5) ＜函数体＞：即大括号{}括起来的部分，这一部分的代码表明了该函数可以实现的功能。

下面是两个合法的函数定义的例子。

【例 5.3】 输出简单图案。

```
void menu()
{
    printf("************************************************\n");
    printf("              欢迎进入函数章节的学习              \n");
    printf("************************************************\n");
}
```

例 5.3 定义了一个 void 类型的无参函数 menu，当自定义函数无须返回值给主调函数时，必须显示说明函数类型为 void，void 不能省略。

当函数 menu 中的三条 printf 语句执行完毕后，工作流程会自动返回至主调函数。

【例 5.4】 定义一个有参函数，求两个整数中的较大值。

```
int max(int n1,int n2)                    /* n1、n2 是函数 max 的形式参数 */
{
    int result;                           /* 定义变量 result 存储两数中的较大值 */
    result=n1>n2?n1:n2;                   /* 条件表达式的结果即两数中的较大值 */
    return result;                        /* 通过 return 语句将结果返回至主调函数 */
}
```

例 5.4 定义的 max 函数的类型为整型 int，int 类型的函数定义时，可以省略 int 不写。函数类型不为 void 的函数，其函数体内至少有一条 return 语句。当执行至 return 语句时，工作流程将返回到该 return 语句所处函数的主调函数中。同时，return 后所跟的值将被带回到主调函数。

return 是关键字，其常用语法形式：

```
return 表达式;
return (表达式);
```

【例 5.5】 定义一个有参函数用于返回两个任意整数的和。

```
int sum(int a,int b)
```

```
{
    int s;
    s=a+b;
    return s;
}
```

函数定义中,每个形参都必须用一个类型说明符单独说明,不可共用。

到现在为止,已经了解到每一个 C 程序都是若干个函数的集合体,而每一个函数又都有特定的功能,如果将一个 C 程序看作是一个政府机关,那么组成这个程序的若干函数就好比是这个政府机关的各个职能办公室,虽然都有自己独立的任务要完成,但归根究底还是为了整个政府机关的正常运转,有一个共同的大目标。

5.1.3 案例分析:打印图案

问题描述

定义一个函数 print,打印如下结果:

```
1    2    3    4    5    6    7    8    9
2    4    6    8   10   12   14   16   18
3    6    7   12   15   18   21   24   27
4    8   12   16   20   24   28   32   36
```

要点解析

一个函数是由函数头和函数体两部分构成的,从案例功能描述可以确定该函数是一个无返回值的函数,其函数头如下:

```
void print()
```

函数体是函数实现功能的部分,其中书写的代码必须能够执行“规定任务”——打印如上所示四行九列的矩阵。观察矩阵的特点,可以使用双重 for 循环来实现。

程序与注释

```
void print()                          /*函数头:定义无参无返回值函数 print*/
{
    int i;                            /*定义外重循环变量 i*/
    int j;                            /*定义内重循环变量 j*/
    for(i=1;i<=4;i++)                 /*外重循环控制行间的变化*/
    {
        for(j=i;j<=9*i;j+=i)          /*内重循环控制列间的变化*/
        {
            printf("%4d",j);          /*%4d:使用格式修饰符控制列与列间的间距*/
        }
        printf("\n");
    }
}
```

分析与思考

案例中的数字矩阵图案是一个典型的二维图形，可以判断该图案的实现需要双重循环：外循环控制行的变化，每一行的第一个数值等于当次的循环次数，因此确定外循环变量 i 的初始值为 1，终止值为 4；内循环控制每一行列值的变化，后一列的值等于前一列的值与当前行号之和，最后一列的数值等于当次循环次数的 9 倍，因此确定内重循环变量 j 的初始值为 i，终止值为 9 * i，步长为 i。

5.2 函数的调用

5.2.1 函数调用的一般形式

在前面的章节中，我们已经了解到一个 C 程序可以由若干个自定义函数构成，而自定义函数的执行是由主调函数通过函数调用语句实现的，在例 5.2 中，主函数就是主调函数，menu 函数、welcome 函数和 getch 函数就是被调函数。函数调用的一般格式为：

```
<函数名>(参数表);
```

按照被调函数在主调函数中的作用，函数的调用方式可以表现为以下三种形式。

1. 作为单独的函数调用语句

被调函数在主调函数中，以一条独立语句的形式出现，完成一种操作，不带回返回值。

【例 5.6】 分析函数调用过程。

```
#include<stdio.h>
void pattern( )                    /* 无参函数的定义：定义的 pattern 函数的形参列表为空 */
{
    printf("*****\n");
}
void main( )
{
    int i;

    for(i=1;i<=5;i++)
    {
        printf("%d",i);
        pattern( );                /* 函数调用语句：通过函数名调用无参函数 pattern */
    }
}
```

程序运行结果如下：

```
1*****
2*****
```

```
3*****
4*****
5*****
```

自定义函数 pattern 是一个无参无返回值的函数，在主函数中通过一条单独的函数调用语句来调用 pattern 函数，pattern 函数只是完成一种操作，不带回返回值。

2. 作为函数的部分参数

函数的调用结果进一步做其他函数调用的参数，这种函数有返回值。

【例 5.7】 编写一个程序，在自定义函数中实现 3 个数中求最大值，在主函数中输出最后结果。

```
#include<stdio.h>
int max(int a,int b)                    /*两个数求最大值*/
{
    int t;
    t=(a>b?a:b);
    return t;                           /*返回语句：通过 return 返回变量 t 的值*/
}
void main( )
{
    int x1,x2,x3;
    printf("input 3 number:");
    scanf("%d%d%d",&x1,&x2,&x3);
    printf("the number=%d\n",max(x1,max(x2,x3)));
                                        /*函数调用作为输出参数*/
}
```

在主函数中两次出现 max 函数调用，其中 max(x2,x3)是将函数的返回值作为第二次调用 max 函数的其中一个实参。

3. 作为表达式的一部分

将函数的调用结果作为表达式的一部分。

【例 5.8】 调用库函数 pow(a,b)，pow(a,b)的功能是求 a 的 b 次方的值。

```
#include<stdio.h>
#include<math.h>
void main( )
{
    int a=2,b=3,i=4,j=5,c;
    c=pow(a,i)+pow(b,j);                /*两次调用函数 pow,将其结果作为加数和被加数*/
}
```

使用库函数应该有相应的头文件说明，pow 函数的相关说明在 math.h 文件中，因此该程序开头包括了预处理命令#include＜math.h＞。在主函数中可以看到 pow(a,i)和 pow(b,j)作为“＋”运算符的运算分量参加了运算，实际上，任何表达式可以出现的地方

都可以出现函数调用。

5.2.2 函数的参数

前一小节中，初步认识到形参和实参的差别：函数定义时，出现在函数名后括号中的参数是形参；函数调用时，出现在函数名后括号中的参数是实参。函数调用时，实参的数量要与形参的数量一致，多个实参的数据类型要与形参的数据类型一致，多个实参出现的意义顺序要与形参的意义顺序一致。形参和实参的功能是作数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

函数的形参和实参具有以下特点：

(1) 形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。函数调用结束返回主调函数后则不能再使用该形参变量。

(2) 实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此应预先用赋值，输入等办法使实参获得确定值。

(3) 实参和形参在数量上、类型上、顺序上应严格一致，否则会发生类型不匹配的错误。

(4) 函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

【例 5.9】 交换两个变量的值。

```
#include<stdio.h>
void swap(int x,int y)
{
    int t;
    t=x;
    x=y;
    y=t;
}
void main()
{
    int a,b;
    a=2;
    b=3;
    swap(a,b);
    printf("互换失败：a=%d,b=%d\n",a,b);
    getch();
}
```

程序运行结果如下：

```
互换失败：a=2,b=3
```

程序首先定义了一个 swap()函数，它有两个形参 x 与 y。在 main()函数中，定义了两个整型变量 a 和 b，并将 2 和 3 分别赋值给 a 和 b，然后是一个调用语句 swap(a,b);，这里 a,b 是实参。

如图 5-1 所示，在函数调用时，为形参变量 x 与 y 分配内存单元，并将实参 a 和 b 的值传递给形参 x 和 y。在函数 swap()内 x 和 y 的值确实做了交换，由于函数调用结束时，要释放为 x 与 y 所分配的内存单元。换句话说，形参 x 和 y 只在函数 swap()内部有效。可见实参的值不随形参的变化而变化。

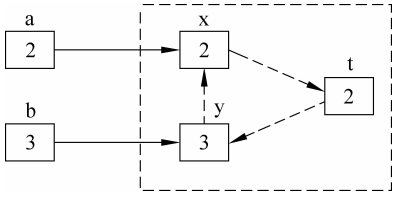

图 5-1　变量变换

5.2.3　函数的说明

C 语言可以由若干个文件组成，每一个文件又可以单独编译，因此当编译程序中的函数调用时，如果不知道该函数参数的个数和类型，编译系统就无法检查形参和实参是否匹配。为了保证函数调用时，编译程序能检查出形参和实参是否满足类型相同，个数相等，并由此决定是否进行类型转换，必须为编译程序提供所用函数的返回值类型和参数的类型、个数，以保证函数调用成功，因此有时候在主调函数中需要对被调函数进行说明。

在前面的例子中，我们总是先写被调用函数，再写主调函数，实际上组成一个程序的多个函数的放置位置是任意的，也就是说，可以先写主调函数，再写被调函数，但需要有相应的函数说明语句。函数说明的方式如下：

```
<函数类型><函数名>(参数类型列表);
```

如：

```
int sum(int a,int b);
```

或者

```
int sum(int,int);
```

应当注意的是：函数的说明和函数的定义在书写形式上比较类似，但两者在本质上是不同的，主要区别如下：

(1) 函数的定义由两部分组成：函数首部和函数体。函数的定义是根据要完成的功能编写一段程序，应有函数具体的功能语句，即函数体；而函数的说明只是对编译系统的一个说明，没有具体的执行动作。

(2) 程序中，函数的定义只能有一次，而函数的说明可以有多次，调用几次函数，就应在各个主调函数中各自做说明。如：

```
void main( )
```

```
{
    int sum(int,int);              /* 主函数中声明 sum 函数 */
  …
}
void fun( )
{
    int sum(int a,int b);          /* fun 函数中声明 sum 函数 */
  …
}
int sum(int a,int b)               /* 定义 sum 函数 */
{
  …
}
```

在上面的程序段中，自定义函数 sum 分别被 main 函数和 fun 函数调用，因而在 main 函数和 fun 函数中各自出现了对被调函数 sum 的函数说明语句，而函数 sum 的定义体只有一个。

观察前面的例子发现：有的主调函数中有被调函数的说明语句，而有的主调函数中确没有对被调函数进行说明。通常在下列两种情况下，可以省略函数声明：当函数的返回值为整型或字符型时，且在同一个文件中既定义函数，又调用该函数；当被调用函数的返回值不是整型或字符型，而是其他类型（如 float，double，void 等），如果函数定义和函数调用在同一个文件中，且函数定义在源程序中的位置在主调函数之前（如例 5.7）。

【例 5.10】 计算两个整数的和，并输出结果。

```
#include<stdio.h>
void main( )
{
    int x,y,s;
    scanf("%d%d",&x,&y);
    s=add(x,y);
    printf("result=%d\n",s);
}
int add(int a,int b)
{
    int total;
    total=a+b;
    return total;
}
```

程序运行结果如下：

```
34
result=7
```

因为 add 函数是整型函数，所以即使 add 函数的定义在主函数之后，在主函数中仍然可以不加函数说明，直接调用。但是，如果函数定义的位置在主调函数之后，且该函数类