第5章 函 数

CHAPTER

C语言程序由函数构成。函数是完成特定功能的程序段，是程序实现模块化的基本单元。C语言不仅提供了丰富的库函数，还允许用户定义自己的函数。使用函数可以减少重复编写程序的工作量，也可以简化程序调试的难度。

5.1 函数概述

一个C语言程序由一个主函数main()和若干辅助函数构成。通常，不是把一个程序的所有函数都放在一个源程序文件中，而是依据函数的功能将其放在多个文件中。这样一来，一个较大的程序就可以由多人同时编写，便于分工合作。由此可知，一个C语言程序由一个或多个源程序文件构成，每个源程序文件由一个或多个函数构成，每个函数由若干行程序语句组成。函数是程序最基本的构成单位，语句是程序的最小构成单位。

在C语言中，包含主函数main()在内的所有函数之间的关系是平等的，即C语言中的函数可以按照任何顺序出现在C程序中，函数之间没有包含与被包含的关系，只有调用与被调用的关系。

注意：一个C语言程序有且只有一个主函数main()，程序的执行从主函数main()开始，最终在主函数main()结束整个程序的运行；主函数main()可以调用辅助函数，但辅助函数不能调用主函数。

从用户使用的角度来看，函数分为两类。

(1) 标准函数(也称库函数或系统函数)。由系统提供，是一些常用功能模块的集合。每个标准函数都已经被设计好以完成某种功能，如函数scanf()和printf()分别完成数据的输入和输出功能。用户若想使用这些功能，就不必再编写代码了，只需要将这个函数所在的头文件用#include宏命令包含进程序中即可。值得注意的是，每个C版本所提供的系统函数的功能和数量都不尽相同，使用时需要查看相应的函数说明。

(2) 用户自定义函数。系统函数是不可能把所有功能都考虑进去的，

一个项目中的绝大多数功能都需要用户自己编写代码,这也是 C 语言中体现用户编程能力的重要一环,也是编程人员必须掌握的基本能力。

从函数的形式来看,函数分为两类。

(1) 无参函数。例如,getchar()是系统无参函数,例 5.1 中的 printmessage()和 printstar()是用户自定义的无参函数。在调用无参函数时,调用函数并不会将数据传递给被调用函数。

(2) 有参函数。例如,putchar()是系统有参函数,例 5.2 中的 area()是用户自定义的有参函数。在调用有参函数时,调用函数和被调用函数之间有数据传递。也就是说,调用函数将数据传递给被调用函数使用,被调用函数中的数据也可以被带回到调用函数中使用。

下面通过两个程序简单地看一看函数的作用。

【例 5.1】 简单函数定义和调用示例。

```
#include "stdio.h"
void printstar()
{ printf("\t********** \n");
}
void printmessage()
{ printf("\t * Welcome * \n");
}
void main()
{ printstar();
  printmessage();
  printstar();
}
```

【运行结果】

```
**********
* Welcome *
**********
```

例 5.1 中的 printstar()和 printmessage()函数都是无参函数,无返回值。

【例 5.2】 输入半径值,计算圆的面积。

```
#include "stdio.h"
float area(float x)
{ float s;
  s=3.14159 * x * x;
  return (s);
}
void main()
{ float r,s;
  printf("Enter r:"); scanf("%f",&r);
```

```
    s=area(r);
    printf("s=%.3f\n",s);
}
```

【运行结果】

```
Enter r:10↙
s=314.159
```

例 5.2 中的 area()函数是有参函数，被放在了主函数之前。是否可以将它放在主函数之后？这个问题将在 5.4 节中讨论。

5.2　函数的定义

函数的定义就是确定一个函数需要完成什么样的功能以及如何运行。函数定义的一般形式为

```
[函数存储类别][函数返回值类型] 函数名([函数形式参数表])
{ 函数体说明部分
  函数功能语句序列
  [return 表达式;]
}
```

说明：

(1) 函数存储类别：表明该函数是外部函数还是内部函数。

若省略函数存储类别，则系统默认为 extern，表明该函数是一个外部函数，可以被程序中的其他函数调用。若是 static 标识符，则表明该函数是内部函数，只能在定义该函数的文件中被调用。

(2) 函数返回值类型：表明调用该函数时将带回一个何种类型的值。

函数可以通过 return 语句带回一个确定的值。若定义函数时省略了函数返回值类型，则系统默认为 int；若函数返回值类型为 void，则说明函数没有返回值。

(3) 函数名：代表该函数的一个标识符，也是该函数的入口地址(指针常量)。

使用函数时一定要通过函数名称进行标识。该名称要符合标识符的起名规则，虽然可以随意起名，但要尽量做到见名知义。

(4) 函数形式参数表：函数与其他函数相联系的一种桥梁和纽带。解决问题时，已知的数据信息可以通过形式参数表传入函数，以完成函数指定的功能。

函数形式参数表的一般格式为

```
类型 1　参数 1,类型 2　参数 2,…
```

此时，每个参数都是变量，必须分别说明其类型，即使参数变量的类型相同，也必须分别说明，例如

```
int add(int x,int y)                /* 函数头 */
```

```
{ int c;                                /* 两个花括号之间的是函数体 */
  c=x+y;
  return c;
}
```

函数形参的类型也可以统一在函数头下方说明,例如

```
int add(x,y)                            /* 函数头 */
int x,y;                                /* 形式参数变量类型说明 */
{ int c;
  c=x+y;
  return c;
}
```

(5) 函数体说明部分:定义和说明一些实现函数功能的变量和类型等,如上面的变量 c 的定义。

(6) 函数功能语句序列:实现函数功能的语句的有机集合。

(7) return 表达式:函数运行结束并将表达式值带回到调用函数。

这是函数的主体,只有按照功能编写相应的语句行,最终才能实现整个程序的功能。

5.3 函数调用

5.3.1 函数调用的一般形式

函数只有被调用才能被真正执行。调用函数是指通过函数名把指定的参数传送到函数中,使得该函数带有指定的参数值以完成具体的函数功能。函数调用的一般形式为

函数名([实际参数表])

如果被调用的函数是无参函数,则没有“实际参数表”,但是圆括号不能省略。如果“实际参数表”包括两个或两个以上的实际参数,则实际参数之间用逗号隔开。另外,实际参数的类型名不能写上,实际参数的个数应与形式参数的个数相同,且相对应的实际参数和形式参数的类型也要一致。

5.3.2 函数调用的方式

按照函数在程序中出现的位置,函数调用有以下三种方式。

1. 函数语句

把函数调用作为一个语句,不要求函数有返回值,只要求函数完成一定的操作,例如例 5.1 中的语句

```
printmessage();
```

2. 函数表达式

函数调用出现在一个表达式中,要求函数带回一个确定的值以参加表达式的运算,例如

```
m=add(a,b)/2
```

其中，函数 add(a,b)是表达式的一部分，它的值除以 2 再赋给变量 m。

3. 函数参数

函数调用作为一个函数的参数，实质是将函数的返回值作为这个函数的一个实参，例如

```
m=add(add(a,b),c);
```

其中，add(a,b)是一次函数调用，它的值作为函数 add 另一次调用的实参，m 的值是 a、b、c 三者之和。又如

```
printf("%d\n",add(a,b));
```

把 add(a,b)作为 printf 函数的一个实参。

【例 5.3】 通过调用函数计算任意 3 个整数的和。

```
#include "stdio.h"
int add(int x,int y,int z)
{ return x+y+z;
}
void main()
{ int a,b,c;
  printf("Input a,b&c:");
  scanf("%d%d%d",&a,&b,&c);
  printf("add=%d\n",add(a,b,c));
}
```

【运行结果】

```
Input a,b&c:4 5 6↙
add=15
```

5.4　函数引用说明

一个函数在被另一个函数调用时，被调用函数必须存在。另外，如果被调用函数是系统函数，则应在源程序文件的开头用 #include 命令将被调用的库函数所在的头文件(扩展名为 h)包含到本文件中。如果被调用函数是程序中的用户自定义函数，则应在调用函数的说明部分加上引用说明。引用说明是指对函数的名称、返回值类型以及形参的类型和顺序进行说明。引用说明的主要作用是在程序编译阶段对被调用函数的合法性进行全面检查，如函数名是否正确，函数返回值的类型与 return 语句中的表达式的类型是否相同，形式参数变量与实际参数表达式的类型和个数是否一致等。函数引用说明的形式为

```
函数返回值类型 函数名(类型 1 形参 1,类型 2 形参 2,…);
```

其中，形参名可以省略，又写成

函数返回值类型 函数名(类型 1,类型 2,…);

当参数类型都为 int 或 char 时,类型名也可以省略,又写成

函数返回值类型 函数名();

不能再省略了,尤其是一对圆括号和末尾的分号绝对不能省略。

例如,例 5.3 中的主函数 main()应该修改为

```
void main()
{ int a,b,c;
  int add(int x,int y,int z);                    /* 函数引用说明 */
  printf("Input a,b&c:");
  scanf("%d%d%d",&a,&b,&c);
  printf("add=%d\n",add(a,b,c));                 /* 函数调用 */
}
```

那么,为什么例 5.3 在没有函数引用说明的情况下也能正确执行呢? 其实,下列几种情况可以省略函数引用说明。

(1) 函数的定义写在主调函数之前。

(2) 函数的返回值类型为 int 或 char(省略类型默认为 int)。

(3) 在所有函数定义的前面已经添加了函数引用说明。

例 5.3 既符合(1),也符合(2),所以可以省略函数引用说明。

另外,请读者注意函数定义与函数引用说明的区别。

5.5 函数的参数和返回值

5.5.1 形式参数和实际参数

定义函数时的参数称为形式参数,简称形参。调用函数时的参数称为实际参数,简称实参。在调用有参函数时,首先应进行参数传递。参数传递的规则是:实参表达式的值依次对应地传递给形参表中的各个形参变量。这种从实参到形参的传递是单向的值传递。确切地说,在函数被调用时,系统为每个形参变量分配了存储单元,并把相应的值传递到这些存储单元作为形参变量的初值,然后执行规定的操作。这种传值操作的特点是:函数中对形参的操作不会影响主调函数中的实参值,即形参变量的值不能传回给实参表达式。

【例 5.4】 函数调用参数传递举例。

```
#include "stdio.h"
void swap(int x,int y)        /* 函数定义 */
{ int t;
  t=x;x=y;y=t;
  printf("x=%d, y=%d\n",x,y);
}
```

```
void main()
{ int a=10,b=20;
  printf("a=%d, b=%d\n",a,b);
  swap(a,b);                    /* 函数调用 */
  printf("a=%d, b=%d\n",a,b);
}
```

【运行结果】

```
a=10,b=20
x=20,y=10
a=10,b=20
```

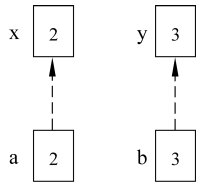

图 5.1　例 5.4 参数传递方式

函数 swap()的功能是交换形参变量 x、y 的值。函数调用开始，为形参变量 x、y 分配存储单元，同时进行参数传递，参数传递方式见图 5.1。函数 swap()中对形参变量 x、y 值的修改不会影响实参 a、b 的值。

说明：

(1) 在函数调用之前，形参变量并未被分配内存单元。只有在发生函数调用时，形参表中的各个变量才会被分配内存单元。函数调用结束后，形参变量所占用的内存单元也会被释放。

(2) 形参是变量，实参是表达式(常量和变量是特殊的表达式)。例如，对于例 5.3 中的函数 add()，以下调用都是正确的。

```
add(10,5,3);   add(a,b,c);   add(a+10,b-5,c*2);   add(10,a*b,b-c);
```

(3) 参数传递是赋值运算，即只能由实参传给形参，而不能由形参传回给实参。在内存中，实参和对应的形参分别占用不同的存储单元。

(4) 形参变量与实参表达式的类型需要一致，否则系统会按照赋值运算规则自动进行类型转换，即将实参表达式的值转换成对应的形参变量类型后再传递给形参变量。

5.5.2　函数的返回值

在大多数情况下，人们希望通过函数调用返回一个有确定意义的值，这个值就是函数的返回值。C 语言中，通过 return 语句可以将被调用函数中的一个确定值带回到调用函数中，return 语句的格式为

return 表达式;

或

return (表达式);

return 语句的功能是：从函数中退出，返回到主调函数中的调用处，同时带回表达式的值。

说明：

(1) 一次函数调用只能得到一个返回值。一个函数中可以有多个 return 语句，但只有第一个被执行的 return 语句起作用。

(2) 函数返回值的类型是定义函数时指定的类型。若 return 语句中的表达式的值的类型与函数返回值的类型不一致，则以函数返回值的类型为准，对数值型数据自动进行类型转换。

(3) 若被调函数中没有 return 语句，且函数返回值类型不是 void，则函数并非不带回值，而是带回了一个不确定的无用值。例如，在例 5.4 中，若 swap()函数省略了函数返回值类型 void，并将主函数 main()修改为

```
void main()
{ int a=10,b=20,c;
  printf("a=%d, b=%d\n",a,b);
  c=swap(a,b);
  printf("a=%d, b=%d\n",a,b);
  printf("c=%d\n",c);
}
```

则运行后除了得到例 5.4 的结果外，还会输出 c=11。

(4) 若明确表示函数调用不带回值，则在定义函数时应将类型指定为 void。这样一来，系统就会保证不使函数带回任何值，即不允许在函数中使用 return 语句返回值。在例 5.4 中，因为函数 swap()的返回值类型为 void，所以下面的用法是错误的。

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

编译时会给出出错信息。

5.5.3 指针作为函数参数

1. 普通变量的指针与函数参数

要想将普通变量的地址传递给形参变量，则形参变量必须是指针类型。指针作为函数参数进行传递，实质上还是值的单向传递，传递过来的值只不过是一个地址，实参和形参这两个量将指向同一个存储单元。在函数中对形参变量所指向的内存单元的值的改变相当于改变实参所指向的内存单元的值。严格地说，并非改变了实参和形参指针变量的值，而是改变了两个指针所指向的同一个内存单元中的值。在程序设计过程中，程序员往往利用这一点，在编写程序时通过一次函数调用带回多个值。

【例 5.5】 编写程序，通过函数调用方式交换变量 a 和 b 的值。

```
#include "stdio.h"
void swapab(int *x,int *y)
{ int z;
  z=*x; *x=*y; *y=z;
}
void main()
```

```
{ int a=10,b=20;
  printf("Before swap:a=%d,b=%d\n",a,b);
  swapab(&a,&b);
  printf("After swap: a=%d,b=%d\n",a,b);
}
```

【运行结果】

```
Before swap:a=10,b=20
After swap: a=20,b=10
```

函数调用开始，为形参指针变量 x、y 分配存储单元，同时进行参数传递，参数传递方式见图 5.2。此时，*x 就是 a，*y 就是 b，交换 *x 和 *y 的值就是交换变量 a 和 b 的值。

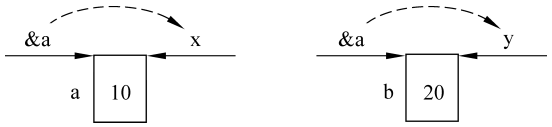

图 5.2　例 5.5 参数传递方式

思考题：若将函数 swapab()写成如下形式，那么还能实现以上功能吗？

```
void swapab(int *x,int *y)
{ int *z;
  z=x; x=y; y=z;
}
```

若改成如下形式，那么还能实现以上功能吗？

```
void swapab(int *x,int *y)
{ int *z;
  *z=*x; *x=*y; *y=*z;
}
```

2. 数组地址与函数参数

前面讨论了数组，数组是存储数据的重要工具。因为数组中存放的数据有先后的次序关系，所以很容易对其进行统一处理。函数是构成程序的基本单位，它可以通过参数传递处理数组。在参数传递中，可以把实参数组的地址直接传递给形参指针变量，然后在函数中处理数组元素，形参指针变量将指向实参数组元素。

(1)一维数组作函数参数。

【例 5.6】 将一维数组中的第一个元素与最后一个元素交换位置。

```
#include "stdio.h"
#define N 8
void exchange(int x[N],int n)
{ int t;
  t=x[0]; x[0]=x[n-1]; x[n-1]=t;
```

```
}
main()
{ int a[N],i;
  for(i=0;i<N;i++)
    scanf("%d",a+i);
  exchange(a,N);
  for(i=0;i<N;i++)
    printf("%d\t",a[i]);
  printf("\n");
}
```

【运行结果】

```
1 2 3 4 5 6 7 8↙
8   2   3   4   5   6   7   1
```

编译时,编译系统自动将函数 exchange()的形参说明 int x[N]转换为 int * x。函数调用开始,为形参 x 分配存储单元,同时进行参数传递,参数传递方式见图 5.3。此时,x[i] 与 a[i](0≤i≤N−1)表示同一数组元素,对形参数组 x 进行操作等同于对实参数组 a 进行操作。

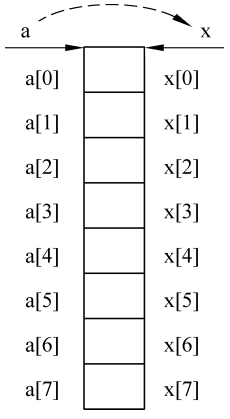

图 5.3 例 5.6 参数传递方式

函数 exchange()中的形参说明 int x[N]也可以写成 int x[]或 int * x,编程时常用这两种形式。

思考题:为什么编译系统会自动将函数 exchange()的形参说明 int x[N]转换为 int * x?

(2) 二维数组作函数参数。

【例 5.7】 将 3×4 数组中的最大值与最小值互换位置。

```
#include "stdio.h"
#define  M  3
#define  N  4
void exchangemm(int x[M][N])
{ int i,j,max,min,hi,hj,li,lj,t;
  max=min=x[0][0];
  hi=hj=li=lj=0;
  for(i=0;i<M;i++)
    for(j=0;j<N;j++)
    { if(x[i][j]>max) { max=x[i][j];hi=i;hj=j; }
      if(x[i][j]<min) { min=x[i][j];li=i;lj=j; }
    }
  t=x[hi][hj]; x[hi][hj]=x[li][lj]; x[li][lj]=t;
}
void main()
{ int a[M][N],i,j;
```