第5章 函 数

CHAPTER

5.1 函数概述

一个C程序一般都是由一个主函数和若干个辅助函数构成的。在C语言中，函数是程序的最基本的执行单位。一个C程序若没有函数就无法执行，也就不能称其为C程序了。作为用语言编写的一个项目，通常也不把所有的函数都放在一个源程序文件中，而是依据所实现的功能将其放在多个文件中，可以由多人共同完成一个项目。这样的分工合作更便于一个项目的完成。

所以说，一个C语言源程序可由多个源程序文件构成，每个文件可由多个完成不同功能的函数构成，每个函数由若干行程序语句组成。在这里，函数是程序的基本构成单位，语句是程序的最小构成单位。

需要注意的是，一个C程序有且只有一个主函数main()，程序执行都是从主函数开始的。也就是说，只有找到这个主函数，程序才能正常执行。包含主函数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，只有主函数与辅助函数及辅助函数之间的调用与被调用的关系。但是，一个函数定义不允许重复出现在一个C程序中。

从用户的使用角度来看，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完成某种功能，如printf()和scanf()分别完成数据的输入和输出功能。用户若想用这些功能，就不必再编写代码了，只需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最重要的一环，也正是编程人员必须要掌握的基本能力。

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，调用函数并不将数据传递给被调用函数。无参函数可以带回函数值，但多数情况下不带回函数值。

（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带回到调用函数使用。

下面，通过两个程序来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】

```
***********
 * Welcom  *
***********
```

此例中的printstar()和printmessage()两个函数都是无参函数，无返回值。将两个函数的位置放在主函数之前，运行结果也是一样的。

【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
```

本例中的 area()函数是有参函数，放在了主函数之前。是否可将它放在主函数之后，这个问题将在 5.4 节中介绍。

5.2　函数定义

函数的定义就是要确定一个函数完成什么样的功能以及怎样运行。函数定义的一般形式如下：

```
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
}
```

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。

若是 extern 标识符，则表明该函数是外部函数，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。若省略函数的存储类别，则系统默认的存储类别为 extern。

(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。

函数通过 return 语句返回值，return 语句中的表达式类型一定要与函数返回值类型一致。若定义函数时省略函数返回值类型，则系统默认为 int。值得注意的是返回值类型为 void 的函数没有返回值。

(3) 函数名：是代表该函数的标识符。

使用函数的时候一定要通过函数名称标识。该名称要符合标识符的取名规则。虽然可随意起名，但也要尽量做到见名知义。

(4) 函数形式参数表：是函数与其他函数联系的桥梁和纽带。函数外部信息可以通过形式参数表传入函数体内，完成函数指定的功能。

函数形式参数表的格式为：

```
类型 1　参数 1,类型 2　参数 2,…
```

每个参数都必须独立说明其类型，即使类型相同的参数，也必须独立说明。例如：

```
int add(int x,int y)
{ int c;
  c=x+y;
```

```
  return(c);
}
```

在 Turbo C 中，与下面函数定义是等价的。

```
int add(x,y)
int x,y;
{ int c;
  c=x+y;
  return (c);
}
```

(5) 函数体说明部分：定义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。

这是函数的主体。只有按功能编写相应的语句行，才能最终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真正执行。调用函数就是通过函数名把指定的参数传送到函数中去，使得该函数带着指定的参数值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)

如果被调用函数是无参函数，则没有“实际参数表”，但括号不能省略。如果“实际参数表”包括两个或两个以上实际参数，则参数之间用逗号隔开。另外，参数的类型名不能写上，参数的类型和个数一定要与形式参数表中各个参数被说明的类型和参数个数一一对应、保持一致。

5.3.2 函数调用的方式

按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。这时不要求函数带回值，只要求函数完成一定的操作。例如，例 5.1 中语句：

```
printmessage();
```

2. 函数表达式

函数调用出现在一个表达式中，这时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 m。

3. 函数参数

函数调用作为一个函数的参数,其实质就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三个整数的和。

```
#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,首先被调用函数必须存在。另外,如果被调用函数是系统函数,应在源程序文件开头用 #include 命令将被调用库函数所在的头文件(扩展名为.h)包含到该文件中来。如果被调用函数是用户自定义函数,应在调用函数的说明部分加上引用说明。所谓引用说明,就是对函数的名称、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利用它在程序的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 返回值的类型是否相同,形式参数与实际参数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,写成:

函数返回值类型 函数名(类型 1,类型 2,…);

在 Turbo C 中，当参数类型都为 int 或 char 时，类型名也可以省略，写成：

```
函数返回值类型 函数名();
```

再也不能省略了，尤其最后的一对圆括号和分号绝不能省略。

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，为什么这个程序不加引用说明也能正确执行呢？其实，有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) 在所有函数定义的前面已加了函数引用说明。

(3) 在 Turbo C 中，函数的返回值类型为 int 或 char(省略类型默认为 int)。

例如，在 Turbo C 中，例 5.3 既符合(1)，也符合(3)；在 VC++ 中，符合(1)；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给形参表中的各个形参变量。这种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，系统为每个形参分配存储单元，把对应的实参值传送到这些存储单元作为形参的初值，然后再执行规定的操作。这种传值操作的特点是：函数中对形参的操作，不会影响到调用函数中的实参值，即形参的值不能传回给实参。

【例 5.4】 函数调用参数传递举例。

```
#include "stdio.h"
viod 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
```

说明：

(1) 在函数调用之前，形参不占内存中的存储单元。只有在发生函数调用时，形参才被分配内存单元。函数调用结束后，形参所占的内存单元也被释放。

(2) 形参是变量，实参可以是常量、变量或表达式。例如，要调用例 5.3 中的函数 add()，下面几种形式都是正确的。

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

(3) 形参与实参的类型要一致，否则按不同类型数值的赋值规则自动进行类型转换。

(4) C 语言规定，实参对形参的数据传递是单向值传递，即只由实参传给形参，而不能由形参传回给实参。在内存中，实参与形参分占不同的存储单元。

5.5.2　函数的返回值

在大多情况下，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：从函数中退出，返回到调用它的函数中，同时带回表达式的值。

说明：

(1) 一次函数调用最多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到的 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 语句中表达式值的类型与定义函数时指定的类型不一致，则以定义函数时的类型为准。对数值型数据，自动进行类型转换。

(3) 若被调用函数中没有 return 语句，且函数返回值类型不是 void，函数不是不带回值，而是带回一个不确定的无用值。例如，在例 5.4 中，若省略函数返回值类型 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);
}
```

运行后除了得到例5.4的结果外，还输出：c=11。

(4) 若明确表示函数调用不带回值，则在定义函数时，类型指定为void。这样，系统就保证不使函数带回任何值，即不允许在函数中用return语句来返回值。在例5.4中，如果将函数swap的返回值类型定义为void，则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

将普通变量的地址传递给形参变量，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，传过来的值只不过是个地址，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内存单元的值的改变就相当于改变实参所指向的内存单元的值。严格地说，并不是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，程序员往往利用这点，在编写程序时，通过函数调用返回多个值。

【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
```

思考题：若将函数swapab()写成如下形式，能完成以上功能吗？

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

若改成如下形式呢?

```
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来处理数组。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) 一维数组作为函数参数

【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"
void exchangefl(int * x,int n)
{ int t;
  t= * x; * x=x[n-1]; x[n-1]=t;
}
void main()
{ int a[8],i;
  for(i=0;i<8;i++) scanf("%d",a+i);
  exchangefl(a,8);
  for(i=0;i<8;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
```

将函数 exchangefl()说明部分改成如下形式也是一样的:

```
void exchangefl(int x[],int n)
```

在这里,x 不是数组名,而是指针变量名。它写成数组形式也是允许的。无论写成什么形式,在函数内处理数组分量时,可以使用指针形式(* x),也可以使用数组分量形式(x[n－1])。希望读者注意体会这一点。

（2）二维数组作为函数参数

【例 5.7】 将3×5数组中的最大值与最小值互换位置。

```
#include "stdio.h"
void exchangemm(int x[][5])
{ int i,j,max,min,hi,hj,li,lj,t;
  max=min=x[0][0];hi=hj=li=lj=0;
  for(i=0;i<3;i++)
    for(j=0;j<5;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[3][5],i,j;
  for(i=0;i<3;i++)
    for(j=0;j<5;j++)
      scanf("%d",&a[i][j]);
  exchangemm(a);
  for(i=0;i<3;i++)
  { for(j=0;j<5;j++)
      printf("%d\t",a[i][j]);
    printf("\n");
  }
}
```

【运行结果】

```
22 18 29 17 45↙
30 89 72 56 61↙
87 93 67 25 36↙
22      18      29      93      45
30      89      72      56      61
87      17      67      25      36
```

上面的函数 enchangemm()中的形参也可能写成 int x[3][5]或 int (*x)[5]，实际上形参变量就是数组指针变量(int (*x)[5]。如果函数用指针来完成，可能会简单一些。例如，将函数 exchangemm()改写成：

```
void exchangemm(int x[][5])
{ int *p,*q,*r,t;
  p=q=x[0];
  for(r=x[0]+1;r<x[0]+3*5;r++)
  { if(*r>*p) p=r;
    if(*r<*q) q=r;
  }
```