第 9 章

函数与函数中的变量

◎ 本章教学微视频：17 个　51 分钟

学习指引

函数是 C 语言程序的基本单位，C 语言程序的功能就是靠每一个函数来具体实现的。通过本章的学习，读者能够掌握 C 语言的函数和函数中变量的使用方法。

重点导读

- 了解函数的概念。
- 掌握调用函数的方法。
- 掌握函数的嵌套调用方法。
- 掌握函数的递归调用方法。
- 掌握函数参数的传递方法。
- 掌握内部函数和外部函数的区别和使用方法。
- 掌握全局变量和局部变量的区别和使用方法。

9.1　函数的概述

函数是 C 源程序的基本模块，通过对函数模块的调用实现特定的功能。C 语言不仅提供了极为丰富的库函数，还允许用户建立自己定义的函数。用户可把自己的算法编成一个个相对独立的函数模块，然后用调用的方法来使用函数。可以说 C 程序的全部工作都是由各式各样的函数完成的，所以也把 C 语言称为函数式语言。

9.1.1　函数的概念

一个 C 语言源程序一般是由一个或多个文件组成，一个源程序文件是一个编译单位，而一个源文件又

可以由若干个函数构成，也就是说，函数是 C 语言程序基本的组成单位。每个程序有且只能有一个 main()函数，其他的函数都是子函数。

C 语言程序总是从 main()函数开始执行，完成对其他函数的调用后再返回到 main()函数，最后由 main()函数结束整个程序。

从本节开始，我们将正式进入函数的学习。要正确地使用函数，首先要掌握函数声明和调用的方法。对于函数结果返回以及参数传递等问题的把握程度是决定读者能否深刻理解函数用法的关键。

【例 9-1】函数调用的简单例子。

（1）在 Visual C++ 6.0 中，新建名称为 9-1.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
void printstar()           /*定义函数 printstar()*/
{
   printf("***************");
}
int sum(int a,int b)       /*定义函数 sum()*/
{
   return a+b;             /*通过 return 返回所求结果*/
}
void main()
{
   int x=2,y=3,z;
   printstar();            /*调用函数 printstar()*/
   z=sum(x,y);             /*调用函数 sum()*/
   printf("\n%d+%d=%d\n",x,y,z);
   printstar();            /*调用函数 printstar()*/
}
```

（3）程序运行结果如图 9-1 所示。

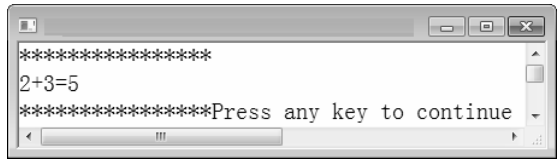

图 9-1　程序运行结果 1

本例是由 3 个函数构成的，分别是 main()、printstar()和 sum()。其中，main()函数是程序的入口函数，是每个 C 语言必须有的函数；printstar()函数是用户自己定义的函数，作用是输出一行星号；而 sum()函数的作用则是用来计算两个数的和，并返回所求结果。在 main()函数中，调用了两次 printstar()函数和一次 sum()函数。

9.1.2　函数的分类

在 C 语言中可从不同的角度对函数分类。

（1）从函数定义的角度看，函数可分为库函数和用户定义函数两种。

库函数：由 C 系统提供，用户无须定义，也不必在程序中做类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。

用户定义函数：由用户按需要写的函数。对于用户定义函数，不仅要在程序中定义函数本身，而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。

（2）从函数是否有返回值的角度看，又可以把函数分为有返回值函数和无返回值函数两种。

（3）从是否需要参数的角度看，又可以把函数分为无参函数和有参函数两种。

无参函数：函数定义、函数说明及函数调用中均不带参数。

有参函数：也称为带参函数。在函数定义及函数说明时都有参数，称为形式参数（简称为形参）。在函数调用时也必须给出参数，称为实际参数（简称为实参）。进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。

尽管 C 语言的函数种类众多，但是在 C 语言中所有的函数定义，包括主函数 main()在内，都是平等的。也就是说，在一个函数的函数体内，不能再定义另一个函数，即不能嵌套定义。但是函数之间允许相互调用，也允许嵌套调用。习惯上把调用者称为主调函数，被调用者称为被调函数。函数还可以自己调用自己，称为递归调用。

9.1.3　函数定义的一般形式

函数作为 C 程序的基本组成部分，是具有相对独立性的程序模块的，能够供其他程序调用，并在执行完自己的功能后，返回调用它的程序中。函数的定义实际上就是描述一个函数所完成功能的具体过程。

由于有些函数需要参数而有些函数不需要参数，所以函数的定义也有两种形式，下面分别进行阐述。

1. 无参函数的定义形式

```
类型标识符 函数名()
{
    声明部分
    执行部分
}
```

例如：

```
int fun()                    //函数首部,声明一个名为 fun 的 int 型函数
{
   int a;                    //声明部分,声明一个 int 型变量 a
   a=8;                      //执行部分,给变量 a 赋值 8
   printf("a=%d\n",a);       //执行部分,输出 a 的值
}
```

（1）类型标识符和函数名所在行称为函数首部，也称函数头。类型标识符则指明了本函数的类型，函数的类型实际上是函数返回值的类型。该类型标识符与前面介绍的各种说明符相同。函数名是由用户定义的标识符，函数名后有一个空括号，其中无参数，但括号必不可少。

（2）{}中的内容称为函数体。在函数体中声明部分在前，它是对函数体内部所用到的变量和常量的类型说明，C 语言要求所有的常量和变量都必须先声明才可以使用。在函数体中执行部分在后，它是完成函

数功能的具体操作语句，必须写在函数体的执行部分。

如果一个函数不需要返回值，此时函数类型标识符可以写为 void。例如：

```
void Hello()
{
   printf("Hello,world \n");
}
```

这里，只把 main 改为 Hello 作为函数名，其余不变。Hello 函数是一个无参函数，当被其他函数调用时，输出"Hello world"字符串。

2. 有参函数定义的一般形式

```
类型标识符 函数名(形式参数表列)
{
   声明部分
   执行部分
}
```

有参函数比无参函数多了一个内容，即形式参数表列。在函数首部的形参表列中给出的参数称为形式参数，它们可以是各种类型的变量，多个参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。形参既然是变量，就必须在形参表中给出形参的类型说明。

例如，定义一个函数，用于求两个数中的较大数：

```
int max(int a, int b)
{
   if(a>b) return a;
   else return b;
}
```

第一行说明 max()函数是一个整型函数，其返回的函数值是一个整数。形参 a，b 均为整型量。a，b 的具体值是由主调函数在调用时传送过来的。在{}中的函数体内，除形参外没有使用其他变量，因此只有执行语句而没有声明部分。在 max()函数体中的 return 语句是把 a 或 b 的值作为函数的值返回给主调函数。有返回值函数中至少应有一个 return 语句。

在 C 语言程序中，一个函数的定义可以放在任意位置，既可放在主函数之前，也可放在主函数之后。

例如，可以把 max()函数置在 main()之后，也可以把它放在 main()之前。

【例 9-2】有参函数的定义和调用。

（1）在 Visual C++ 6.0 中，新建名称为 9-2.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int max(int a,int b)
{
   if(a>b)return a;
   else return b;
}
main()
{
```

```
    int max(int a,int b);
    int x,y,z;
    printf("input two numbers:\n");
    scanf("%d%d",&x,&y);
    z=max(x,y);
    printf("maxmum=%d\n",z);
}
```

（3）程序运行结果如图 9-2 所示。

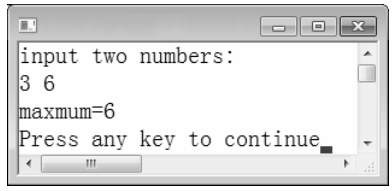

图 9-2　程序运行结果 2

在本例中，可以从函数定义、函数说明及函数调用的角度来分析整个程序，从中进一步了解函数的各种特点。

程序的第 2 行至第 6 行为 max()函数定义。进入主函数后，因为准备调用 max()函数，故先对 max()函数进行说明（程序第 9 行）。从本例可以看出，函数说明与函数定义中的函数首部相同，但是末尾要加分号。程序中语句“z=max(x,y);”为调用 max()函数，并把 x，y 中的值传送给 max()的形参 a，b。max()函数执行的结果（a 或 b）将返回给变量 z，最后由主函数输出 z 的值。

9.2　函数的调用

本节主要介绍函数调用的相关知识。

9.2.1　函数的类型

在定义函数时，必须指明函数的返回值类型，而且 return 语句中表达式的类型应该与函数定义时首部的函数类型是一致的，如果二者不一致，则以函数定义时函数首部的函数类型为准。

【例 9-3】编写函数，计算一个数的立方值。

（1）在 Visual C++ 6.0 中，新建名称为 9-3.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int cube(float x)      /*定义函数 cube(),返回类型为 int*/
{
    float z;           /*定义返回值为 z,类型为 float*/
    z=x*x*x;
    return z;          /*通过 return 返回所求结果*/
}
```

```
void main()
{
   float a;
   int b;
   printf("请输入一个数:");
   scanf("%f",&a);
   b=cube(a);
   printf("%f 的立方为: %d\n",a,b);
}
```

（3）程序运行结果如图 9-3 所示。

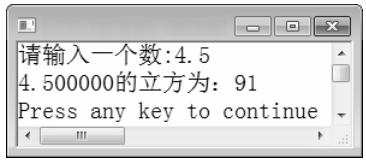

图 9-3　程序运行结果 3

在本例中，函数 cube()定义为整型，而 return 语句中的 z 为实型，二者不一致。按上述规定，若用户输入的数为 4.5，则先将 z 的值转换为整型 91（即去掉小数部分），然后 cube(x)带回一个整型值 91，回到主调函数 main()。如果将 main()函数中的 b 定义成实型，用%f 格式符输出，也是输出 91.0000000。初学者应该做到函数类型与 return 语句返回值的类型一致。

如果一个函数不需要返回值，则将该函数指定为 void 类型，此时函数体内不必使用 return 语句，在调用该函数时，执行到函数末尾就会自动返回主调函数中。

【例 9-4】编写 printdiamond()函数用于输出如图 9-4 所示的图形。

```
**********
 **********
  **********
```

图 9-4　输出图形

（1）在 Visual C++ 6.0 中，新建名称为 9-4.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
void  printdiamond ()       /*定义一个无返回值的函数,返回类型应为 void*/
{
   printf("**********\n");
   printf("**********\n");
   printf("**********\n");
}
void main()
{
   printdiamond();          /*调用 printdiamond()函数*/
}
```

（3）程序运行结果如图 9-5 所示。

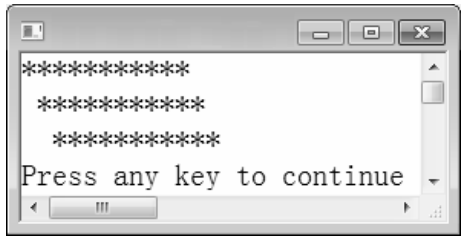

图 9-5　程序运行结果 4

本例中的 printdiamond()函数完成的功能只是输出一个图形，因此不需要返回任何的结果，所以不需要写 return 语句。此时函数的类型使用关键字 void，如果省略不写，系统将认为返回值类型是 int 型。无返回值函数通常用于完成某项特定的处理任务，如打印图形，或输入输出、排序等。

一个函数中可以有一个以上的 return 语句，但不论执行到哪个 return 语句，都将结束函数的调用返回主调函数，即带返回值的函数只能返回一个值。

【例 9-5】编写程序，改写例 9-2，求两个数的较大者。

（1）在 Visual C++ 6.0 中，新建名称为 9-5.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int max(int a,int b)            /*定义函数 max()*/
{
   if(a>b)                      /*如果 a>b,返回 a*/
       return a;
   return b;                    /*否则返回 b*/
}
void main()
{
   int x,y;
   printf("请输入两个整数: ");
   scanf("%d%d",&x,&y);
   printf("%d 和%d 的较大值为: %d\n",x,y,max(x,y));
}
```

（3）程序运行结果如图 9-6 所示。

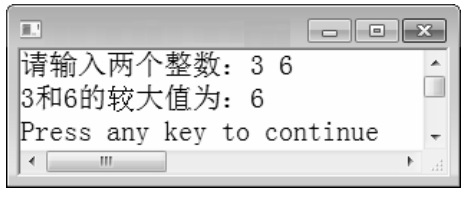

图 9-6　程序运行结果 5

该例使用了两个 return 语句，同样可以求出较大值。在调用 max()函数时，把主调函数中的实参分别传递给形参 x 和 y 后，就执行这个 max()函数。在 max()函数中，定义了一个局部变量 z，然后执行语句“if(x>y)return x; return y;”，其功能是当条件 x>y 成立时执行语句“return x;”返回 x 的值，条件不满足就执行语句“return y;”返回 y。

9.2.2 函数的返回值

当函数被调用时，就会执行函数体内的语句，完成具体的操作。如果希望将这些操作的结果返回给主函数，那么就需要使用函数的返回值。

函数的值只能通过 return 语句返回主函数。

return 语句的一般形式为：

```
return 表达式;
```

或者

```
return (表达式);
```

该语句的功能是计算表达式的值，并将其返回给主函数。

函数的返回值是通过函数中的 return 语句实现的。return 语句将被调用函数中的一个确定值带回主调函数中去。请看下面的实例。

【例 9-6】编写 cube()函数，用于计算 x^3。

（1）在 Visual C++ 6.0 中，新建名称为 9-6.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
long cube(long x)                    /*定义函数 cube(),返回类型为 long*/
{
   long z;
   z=x*x*x;
   return z;                         /*通过 return 返回所求结果,结果类型也应为 long*/
}
void main()
{
   long a,b;
   printf("请输入一个整数:");
   scanf("%ld",&a);
   b=cube(a);
   printf("%ld 的立方为: %ld\n",a,b);
}
```

（3）程序运行结果如图 9-7 所示。

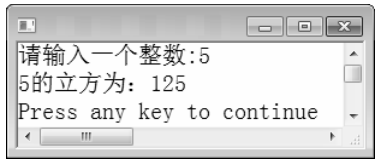

图 9-7　程序运行结果 6

在本例中，程序首先执行主函数 main()，当主函数执行到“c=cube(a);”时调用 cube()函数，把实际参数的值传递给被调用函数中的形参 x。在 cube()函数的函数体中，定义变量 z 得到 x 的立方值，然后通过 return 语句将 z 的值（z 即函数的返回值）返回，返回到调用它的主调函数中，继续执行主函数，将子函数返回的

结果赋给 b，最后输出。

return 语句后面的值也可以是表达式，如例 9-6 中的 cube()函数可以改写为：

```
long cube(long x)
{
   return x*x*x;
}
```

该实例中只有一条 return 语句，后面的表达式已经实现了求 x^3 的功能，先求解后面表达式(x*x*x)的值，然后返回。

根据 return 语句的两种格式可知，例 9-6 中的 return 语句还可以写成：

```
return (z);
```

该语句的执行过程是首先计算表达式的值，然后将计算结果返回给主调函数。

【例 9-7】运行程序，分析当 return 语句中表达式的类型与函数定义时首部的函数类型不一致时，会出现什么结果。

（1）在 Visual C++ 6.0 中，新建名称为 9-7.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int cube(float x)                          /*定义函数 cube(),返回类型为 int*/
{
   float z;                                /*定义返回值为 z,类型为 float*/
   z=x*x*x;
   return z;                               /*通过 return 返回所求结果*/
}
void main()
{
   float a;
   int b;
   printf("请输入一个数:");
   scanf("%f",&a);
   b=cube(a);
   printf("%f 的立方为: %d\n",a,b);
}
```

此程序在 Visual C++ 6.0 环境下编译时出现警告，原因是在 cube()函数中 return 语句后的表达式 z 的数据类型和 cube 函数的类型不一致，但是依然可以运行。

（3）运行结果如图 9-8 所示。

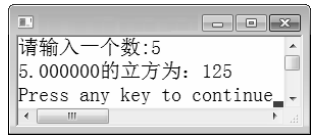

图 9-8　程序运行结果 7

在这个改写过的程序里，函数 cube()被定义为整型，而 return 语句中的 z 为实型，二者不一致。按上述

规定，若用户输入的数为 5.4，则先将 z 的值转换为整型 157（即去掉小数部分），然后 cube(x)带回一个整型值 157 回到主调函数 main()。

下面再来看一个有多个 return 语句的实例。

【例 9-8】求较大值的函数的定义。

（1）在 Visual C++ 6.0 中，新建名称为 9-8.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int max(int a,int b)                    /*定义函数 max()*/
{
   if(a>b)                              /*如果 a>b,返回 a*/
       return a;
   return b;                            /*否则返回 b*/
}
void main()
{
   int x,y;
   printf("请输入两个整数：");
   scanf("%d%d",&x,&y);
   printf("%d 和%d 的较大值为：%d\n",x,y,max(x,y));
}
```

（3）程序运行结果如图 9-9 所示。

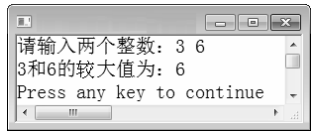

图 9-9　程序运行结果 8

注意：使用 return 语句是无法将多个值返回主调函数中的，每次调用时只能有一个 return 语句被执行，因此返回的只能有一个值。

9.2.3　函数调用的方式

在 C 语言中，程序是通过对函数的调用来执行函数体的。函数调用的一般形式为：

```
函数名([实际参数表])
```

其中，[]中的内容可以省略，说明是对无参函数的调用。实际参数表中的参数可以是常数、变量或其他构造类型数据及表达式，各实参之间用逗号分隔。

函数调用时的说明如下。

（1）调用函数时，函数名称必须与具有该功能的自定义函数名称完全一致。

（2）实参在类型上必须按顺序与形参一一对应和匹配。如果类型不匹配，C 编译程序将按赋值兼容的规则进行转换。如果实参和形参的类型不赋值兼容，通常并不给出出错信息，且程序仍然继续执行，只是得不到正确的结果。

（3）如果实参表中包括多个参数，对实参的求值顺序随系统而异。有的系统按自左向右顺序求实参的值，有的系统则相反。

在 C 语言中，可以用以下几种方式调用函数。

1. 函数表达式

函数作为表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如：

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

就是一个赋值表达式，把函数 max()的返回值赋予变量 z。

2. 函数语句

C 语言中的函数可以只进行某些操作而不返回函数值，这时的函数调用可作为一条独立的语句，即函数调用的一般形式加上分号构成的函数语句。例如：

```
printf("%d",a);
scanf("%d",&b);
```

都是以函数语句的方式调用函数的。

3. 函数实参

函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。例如：

```
printf("%d",max(x,y));
```

即是把 max()函数调用的返回值又作为 printf()函数的实参来使用的。

总的来说，void 类型的函数使用函数语句的形式，因为 void 类型没有返回值；对于其他类型的函数，在调用时一般采用函数表达式的形式。

【例 9-9】编写一个函数，求任意两个整数的最小公倍数。

（1）在 Visual C++ 6.0 中，新建名称为 9-9.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int sct(int m,int n)                    /*定义函数 sct 求最小公倍数*/
{
   int temp,a,b;
   if(m<n)                              /*如果 m<n,交换 m、n 的值,使 m 中存放较大的值*/
   {
       temp=m;
       m=n;
       n=temp;
   }
   a=m;  b=n;                           /*保存 m、n 原来的数值*/
   while(b!=0)                          /*使用辗转相除法求两个数的最大公约数*/
   {
       temp=a%b;
```

```
        a=b;
        b=temp;
    }
    return(m*n/a);                    /*返回两个数的最小公倍数,即两数相乘的积除以最大公约数*/
}
void main()
{
    int x,y,g;
    printf("请输入两个整数: ");
    scanf("%d%d",&x,&y);
    g=sct(x,y);                       /*调用 sct()函数*/
    printf("最小公倍数为: %d\n",g);    /*输出最小公倍数*/
}
```

（3）程序运行结果如图 9-10 所示。

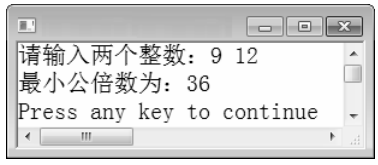

图 9-10　程序运行结果 9

该程序调用了 sct()函数，该函数有两个参数，因此在调用时实参列表也有两个参数，且这两个参数的个数、类型、位置是一一对应的。sct()函数有返回值，因此在主调函数中，函数的调用是参与一定的运算的，这里参与了赋值运算，将函数的返回值赋给了变量 g。

要特别注意，只有实参的个数、类型和顺序与被调用函数所要求的参数个数、类型和顺序一致，才能正确地进行数据传递。

9.2.4　对被调用函数的声明和函数原型

在主函数中调用某函数之前，应该对被调函数进行声明，这与使用变量之前要先进行变量声明是一样的。在主函数中对被调函数进行声明的目的是告知编译系统被调函数返回值的类型，以便在主函数中按照该类型对返回值进行相应的处理。

函数声明的目的是使编译系统在编译阶段对函数的调用进行合法性检查，判断形参与实参的个数、类型及顺序是否匹配。

对被调用函数进行声明的一般形式如下：

```
函数类型  函数名(数据类型 形参 1,数据类型 形参 2,…);
```

例如：

```
int putlll(int x,int y,int z,int color,char *p)      /*声明一个整型函数*/
char *name(void);                                    /*声明一个字符串指针函数*/
void student(int n, char *str);                      /*声明一个不返回值的函数*/
float calculate();                                   /*声明一个浮点型函数*/
```

但是在以下三种情况下，可以省去对被调用函数的说明。

（1）当被调用的函数出现在调用函数之前时。因为在调用之前，编译系统已经知道了被调用函数的函数类型、参数个数、类型和顺序。所以，在主调函数中可以不对被调函数再进行说明而直接调用。

（2）如果在所有函数定义之前，在函数外部（如文件开始处）预先对各个函数进行了说明，则在调用函数中可默认对被调用函数的说明。例如：

```
char str(int a);
float f(float b);
main()
{
    …
}
char str(int a)
{
    …
}
float f(float b)
{
    …
}
```

在这个例子中，最开始处的两行对 str()函数和 f()函数预先进行了说明。因此，在以后各函数中无须对 str()和 f()函数再作说明就可直接调用。

（3）对库函数的调用不需要再进行说明，但必须把该函数的头文件用#include 命令包含在源文件前部。

注意：如果一个函数没有声明就被调用，编译程序并不认为出错，而将此函数默认为整型函数。因此，当一个函数返回其他类型，又没有事先说明，编译时将会出错。

【例 9-10】编写一个函数，求半径为 r 的球的体积。球的半径 r 由用户输入。

（1）在 Visual C++ 6.0 中，新建名称为 9-10.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
double volume(double);              /*函数的声明*/
void main()
{
   double r,v;
   printf("请输入半径: ");
   scanf("%lf",&r);
   v=volume(r);
   printf("体积为: %lf\n\n",v);
}
double volume(double x)
{
   double y;
   y=4.0/3*3.14*x*x*x;
   return y;
}
```

（3）程序运行结果如图 9-11 所示。

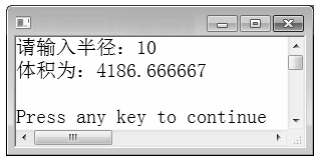

图 9-11　程序运行结果 10

本例中，被调函数 volume()定义在调用之后，需要在调用该函数之前给出函数的声明，声明的格式只需要在函数定义的首部加上分号，且声明中的形参列表只需给出参数的类型即可，参数名称可写可不写，如果有多个参数则用逗号隔开。

9.3　函数的嵌套调用

在 C 语言中，函数之间的关系是平行的、独立的，也就是在函数定义时不能嵌套定义，即一个函数定义的函数体内不能包含另外一个函数的完整定义。但是，C 语言允许在一个函数的定义中出现对另一个函数的调用，这样就出现了函数的嵌套调用。也就是说，在调用一个函数的过程中可以调用另外一个函数。

下面就来看一个函数的嵌套调用实例。

【例 9-11】计算 s=22!+32!。

（1）在 Visual C++ 6.0 中，新建名称为 9-11.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
long f1(int p)
{
   int k;
   long r;
   long f2(int);
   k=p*p;
   r=f2(k);                          /*在 f1()内调用 f2()函数*/
   return r;
}
long f2(int q)
{
   long c=1;
   int i;
   for(i=1;i<=q;i++)
   c=c*i;
   return c;
}
main()
{
   int i;
   long s=0;
   for(i=2;i<=3;i++)
```

```
    s=s+f1(i);                          /*调用 f1()函数*/
    printf("\ns=%ld\n",s);
}
```

（3）程序运行结果如图 9-12 所示。

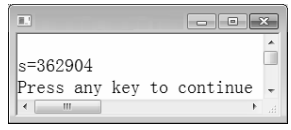

图 9-12　程序运行结果 11

本例编程思路非常简单，可以编写两个函数：一个用来计算平方值的函数 f1()；另一个用来计算阶乘值的函数 f2()。主函数先调用 f1()计算出平方值，再在 f1()中以平方值为实参，调用 f2()计算其阶乘值，然后返回 f1()，再返回主函数，在循环程序中计算累加和。

如例 9-11 中是两层嵌套的情形，其程序的执行顺序如图 9-13 所示。

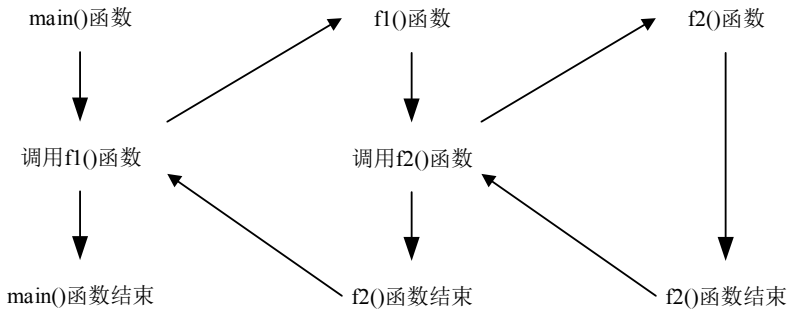

图 9-13　嵌套函数执行顺序

其执行过程如下。

（1）执行 main()函数的函数体部分。

（2）遇到函数调用语句，程序转去执行 f1()函数。

（3）执行 f1()函数的函数体部分。

（4）遇到函数调用 f2()函数，转去执行 f2()函数的函数体。

（5）执行 f2()函数体部分，直到结束。

（6）返回 f1()函数调用 f2()处。

（7）继续执行 f1()函数的尚未执行的部分，直到 f1()函数结束。

（8）返回 main()函数调用 f1()处。

（9）继续执行 main()函数的剩余部分，直到结束。

9.4　函数的递归调用

如果在调用一个函数的过程中，又直接或间接地调用了该函数本身，这种形式称为函数的递归调用，而这个函数就称为递归函数。递归函数分为直接递归和间接递归两种。C 语言的特点之一就在于允许函数

的递归调用。在递归调用中，主调函数又是被调函数。执行递归函数将反复调用其自身，每调用一次就进入新的一层。

直接递归就是函数在处理过程中又直接调用了自己。例如：

```
int func(int a)
{
   int b,c;
   …
   c=func(b);
   …
}
```

其执行过程如图 9-14 所示。

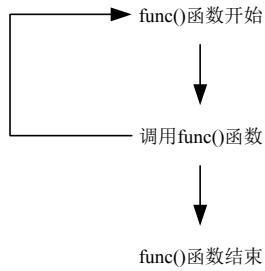

图 9-14　递归函数执行过程

如果函数 p 调用函数 q，而函数 q 反过来又调用函数 p，就称为间接递归。例如：

```
int func1(int a)
{
   int b,c;
   …
   c=func2(b);
   …
}
int func2(int x)
{
   int y,z;
   …
   z=func1(y);
   …
}
```

其执行过程如图 9-15 所示。

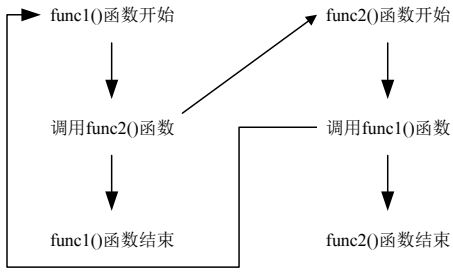

图 9-15　函数的执行过程

以上函数都是递归函数。但是运行这些函数都将无休止地调用其自身，这当然是不正确的。为了防止递归调用无终止地进行，必须在函数内有终止递归调用的手段。常用的办法是加条件判断，满足某种条件后就不再进行递归调用，然后逐层返回。例如，可以用 if 语句来控制只有在某一条件成立时才继续执行递归调用，否则不再继续。下面举例来说明递归调用的执行过程。

【例 9-12】用递归方法求 n!（n>0）。

（1）在 Visual C++ 6.0 中，新建名称为 9-12.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
long fac(int n)                          /*定义求阶乘的函数 fac()*/
{
   long m;
   if(n==1)
   m=1;
   else
   m=fac(n-1)* n;                        /*在函数的定义中又调用了自己*/
   return m;
}
main()
{
   int n; float y;
   printf("input an integer number:\n");
   scanf("%d",&n);
   printf("%d!=%ld\n",n,fac(n));         /*输出 n!*/
}
```

（3）程序运行结果如图 9-16 所示。

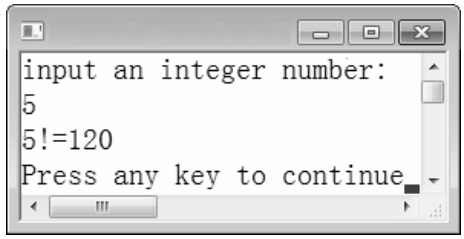

图 9-16　程序运行结果 12

本例采用递归法求解阶乘，就是 5!=4!*5，4!=3!*4，…，1!=1，我们可以用下面的递归公式表示：

$$\begin{cases} n!=1 & (n=0,1) \\ n!=n\times(n-1)! & (n>1) \end{cases}$$

可以看出，当 n>1 时，求 n 的阶乘公式是一样的，因此可以用一个函数来表示上述关系，即 fac() 函数。

main()函数中只调用了一次 fac()函数，整个问题全靠一个 fac(n)函数调用来解决。如果 n 的值为 5，整个函数的调用过程如图 9-17 所示。

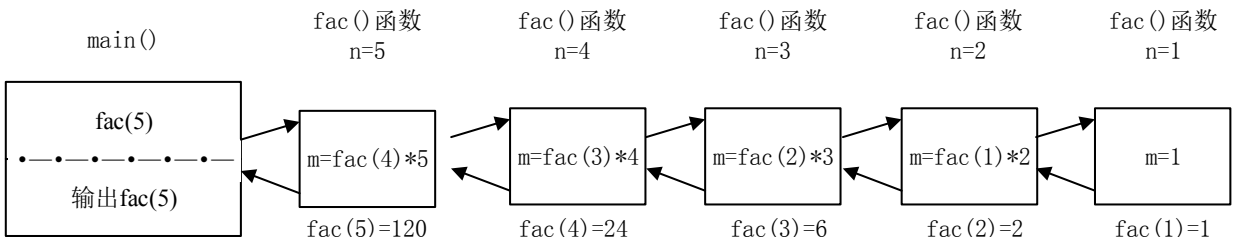

图 9-17　函数的调用过程

从上图可以看出，fac()函数共被调用了 5 次，即 fac(5)、fac(4)、fac(3)、fac(2)、fac(1)。其中，fac(5)是 main()函数调用的，其余 4 次是在 fac()函数中进行的递归调用。在某一次的 fac()函数调用中，并不会立刻得到 fac(n)的值，而是一次次地进行递归调用，直到 fac(1)时才得到一个确定的值，然后再递推出 fac(2)、fac(3)、fac(4)、fac(5)。

在很多情况下，采用递归调用形式可以使程序变得简洁，增加可读性。但很多问题既可以用递归算法解决，也可以用迭代算法或其他算法解决，而后者往往计算的效率更高，更容易理解。如例 9-12 可以用递推法，即从 1 开始乘以 2，再乘以 3…直到 n 来实现。

例如：

```
#include<stdio.h>
long fac(int n)
{
   int i;long m=1;
   for(i=1;i<=n;i++)
   {
       m=m*i;
   }
   return m;
}
main()
{
   int n;
   float y;
   printf("input an integer number:\n");
   scanf("%d",&n);
   printf("%d!=%ld\n",n,fac(n));
}
```

【例 9-13】用递归法求 Fibonacci 数列（斐波那契数列）。

（1）在 Visual C++ 6.0 中，新建名称为 9-13.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
long fibonacci(int n)                /*求 fibonacci 中第 n 个数的值*/
{
   if(n==1||n==2)                    /*fibonacci 数列中前两项均为 1,终止递归的语句*/
      return 1;
   else
```

```
    /*从第 3 项开始,下一项是前两项的和*/
    return(fibonacci(n-1)+fibonacci(n-2));
}
main()
{
    int n,i;
    long y;
    printf("Input n:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)                /*列出 fibonacci 数列的前 n 项*/
    {
        y=fibonacci(i);
        printf("%d ",y);
    }
    printf("\n");
}
```

（3）程序运行结果如图 9-18 所示。

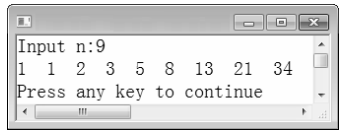

图 9-18　程序运行结果 13

本例仍采用递归方法输出前 n 项的 Fibonacci 数列。Fibonacci 数列的前两项都为 1，从第 3 项开始，每一项都是前两项的和，例如，1，1，2，3，5，8，13，21，35，…可以用下面的公式表示：

$$fibonacci(n)=1 \qquad (n=1,2)$$

$$fibonacci(n)=fibonacci(n-1)+fibonacci(n-2) \qquad (n>2)$$

其中，n 表示第几项，函数值 fibonacci(n)表示第 n 项的值。当 n 的值大于 2 时，每一项的计算方法都一样，因此，可以定义一个函数 f(n)来计算第 n 项的值，递归的终止条件是当 n=1 或 n=2 时。

【例 9-14】Hanoi（汉诺）塔问题。

这是一个典型的只有用递归方法才能解决的问题。

有 3 根针 A、B、C，A 针上有 64 个盘子，盘子大小不等，大的在下，小的在上。图 9-19 所示为汉诺塔模型，要求把这 64 个盘子从 A 针移到 C 针，在移动过程中可以借助 B 针，每次只允许移动一个盘子，且在移动过程中，在 3 根针上都保持大盘在下，小盘在上，要求编程序打印出移动的步骤。

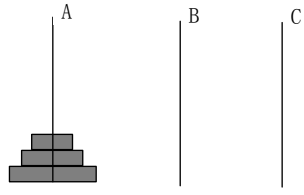

图 9-19　汉诺塔模型

下面先来分析一下 A 针上仅有 3 个盘子时，如何将这 3 个盘子移动到 C 针上。

若想移动最底部的 3 号盘子，则必须移开它上面的 2 号盘子，而若想移动 2 号盘子，又必须移动上面

的 1 号盘子，如图 9-20 所示。

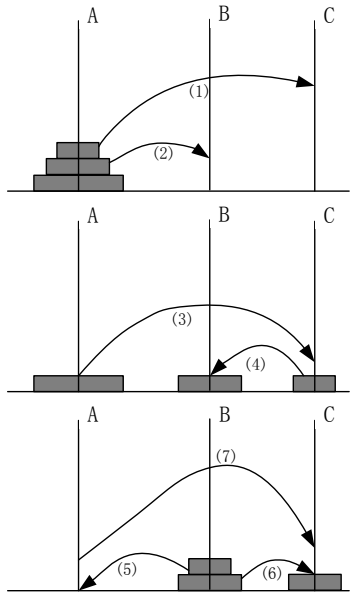

图 9-20　盘子移动过程

首先经过步骤（1）将 1 号盘子从 A 移到 C，再经过步骤（2）将 2 号盘子从 A 移到 B。然后经过步骤（3）将 1 号盘子从 C 移到 B，并经过步骤（4）将 3 号盘子从 A 移到 C。这样 3 号盘子就移动到 C 上了。再考虑将 2 号盘子移动到 C 上。所以经过步骤（5）将 1 号盘子从 B 移到 A，再经过步骤（6）将 2 号盘子从 B 移到 C，最后再经过步骤（7）将 1 号盘子从 A 移到 C，则整个过程就完成了。当盘子数增加时，只要按着这样的递归规则来移动，最初的大问题就会逐渐被分解为规模更小的问题，最终当小问题被逐个击破之后，大问题也就随之解决了。

从上述简单的情况出发将问题推广，可见当盘子的数目为 1 时，只要将盘子从 A 直接移动到 C 上即可。当盘子的数目 n 大于 1 时，则需要利用 B 来辅助。这时需要想办法将 n−1 个较小的圆盘依照规则从 A 移动到 B 上，再将剩下的最大的盘子从 A 移动到 C，最后，再将 n−1 个小盘依照规则从 B 移动到 C。如此下去，n 个圆盘的移动问题就可以分解为两次 n−1 个圆盘的移动问题，也就是分而治之。

由于游戏规则限定每次只能移动一个盘子，且不允许大盘放在小盘上面，所以移动 64 个盘子无疑是一项非常浩大的工程。据估计，解决 64 层汉诺塔问题总共需要移动盘子的次数为将超过 1.8×10^{19}。这是一个天文数字，若每微秒可以计算（并不输出）一次移动，那么求得最终结果也需要几乎一百万年。因此这里也仅能找出问题的解决方法并解决较小 n 值时的汉诺塔问题，用普通计算机来解决 64 层的汉诺塔是不现实的。下面是用 C 语言编写的求解汉诺塔问题的程序。

（1）在 Visual C++ 6.0 中，新建名称为 9-14.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
void printdisk(char x,char y)                    /*定义打印函数*/
{
   printf("%c----->%c\n",x,y);
}
```

```
void hanoi(int n,char a,char b,char c)        /*定义递归函数 hanoi()完成移动*/
{
   if(n==1)                                   /*如果 A 针上的盘子数只剩下最后一个,移到 C 针上*/
       printdisk(a,c);
   else                                       /*如果 A 针上的盘子数多余一个,执行以下语句*/
   {
       hanoi(n-1,a,c,b);                      /*将 A 针上的 n-1 个盘子借助 C 针先移到 B 针上*/
       printdisk(a,c);                        /*将 A 针上剩下的一个盘子移到 C 针上,即打印出来移动方式*/
       hanoi(n-1,b,a,c);                      /*将 n-1 个盘子从 B 针借助 A 针移到 C 针上*/
   }
}
void main()
{
   int n;
   printf("Input n:");
   scanf("%d",&n);                            /*由键盘输入盘子数*/
   hanoi(n,'A','B','C');                      /*调用 hanoi()函数*/
}
```

（3）程序运行结果如图 9-21 所示。

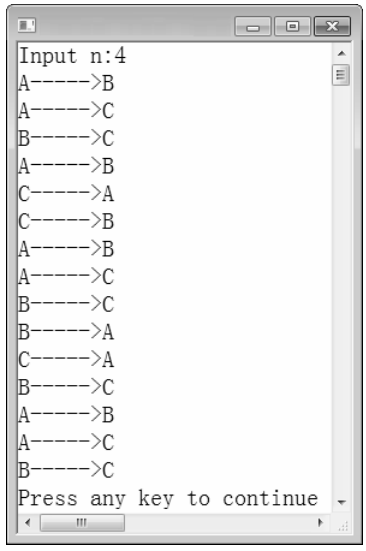

图 9-21　程序运行结果 14

具体来说，将 n 个盘子从 A 针移到 C 针可以分解为以下 3 个步骤。

（1）将 A 针上 n–1 个盘子借助 C 针先移到 B 针上。

（2）把 A 针上剩下的一个盘子移到 C 针上。

（3）将 n–1 个盘子从 B 针借助 A 针移到 C 针上。

这 3 个步骤分成两类操作。

（1）当 n>1 时，将 n–1 个盘子从一个针移到另一个针上，这是一个递归的过程。

（2）将最后一个盘子从一个针上移到另一个针上。

此程序分别用两个函数实现上面的两类操作，用 hanoi()函数实现 n>1 时的操作，用 printf()函数实现一个盘子从一个针上移到另一个针上。

递归作为一种算法，在程序设计语言中被广泛应用，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可以描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。用递归思想写出来的程序往往十分简洁。

注意：递归也有缺点，递归算法解题的运行效率较低。在递归调用的过程中，系统为每一层的返回点、局部量等开辟了栈来存储，系统开销较大。递归次数过多，容易造成栈溢出等问题。总之，在程序中递归算法能不用就不用，能少用就少用。

9.5 函数的参数

本节主要介绍函数参数的相关知识。

9.5.1 函数参数的传递

函数的参数有两类：形式参数（简称形参）和实际参数（简称实参）。函数定义时的参数称为形参，形参在函数未被调用时是没有确定值的，只是形式上的参数。函数调用时使用的参数称为实参。

【例 9-15】将两个数按从小到大排序输出。

（1）在 Visual C++ 6.0 中，新建名称为 9-15.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
void order(int a,int b)                          /*a,b 形式参数*/
{
   int t;
   if(a>b)                                       /*如果 a>b,就执行以下 3 条语句,交换 a,b 的值*/
   {
       t=a;
       a=b;
       b=t;
   }
   printf("从小到大的顺序为:%d  %d\n",a,b);     /*输出交换后的 a,b 的值*/
}
void main()
{
   int x,y;
   printf("请输入两个整数: ");                  /*从键盘输入两个整数*/
   scanf("%d%d",&x,&y);
   order(x,y);                                   /*x,y 是实际参数*/
}
```

（3）程序运行结果如图 9-22 所示。

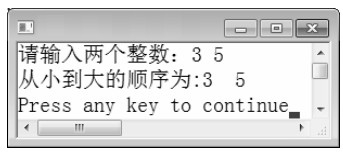

图 9-22 程序运行结果 15

该程序由两个函数 main()和 order()组成，函数 order()定义中的 a 和 b 是形参，在 main()函数中，“order(x,y);”调用子函数，其中的 x，y 是实参。

（1）定义函数时，必须说明形参的类型，如例 9-15 中，形参 x 和 y 的类型都是整型。

注意：形参只能是简单变量或数组，不能是常量或表达式。

（2）函数被调用前，形参不占用内存的存储单元；调用以后，形参才被分配内存单元；函数调用结束后，形参所占用的内存也将被回收，被释放。

（3）实参可以是常量、变量、其他构造数据类型或表达式。例如：

```
order(2,3);            /*实参是常量*/
order(x+y,x-y);        /*实参是表达式*/
```

如果实参是表达式，先计算表达式的值，再将实参的值传递给形参，但要求它有确切的值，因为在调用时是要将实参的值传递给形参的。

（4）实参的个数、出现的顺序和实参的类型，应该与函数定义中形参表的设计一一对应。如例 9-15 中的 order()函数，定义有两个整型的形参，调用它时，实参也要与它对应，而且多个实参之间要用逗号隔开。如果不一致，则会发生“类型不匹配”的错误。

9.5.2　数组元素作为函数参数

数组可以作为函数的参数使用，进行数据传送。数组用作函数参数有两种形式：一种是把数组元素（下标变量）作为实参使用；另一种是把数组名作为函数的形参和实参使用。

数组元素就是下标变量，它与普通变量并无区别。因此它作为函数实参使用与普通变量是完全相同的，在发生函数调用时，把作为实参的数组元素的值传送给形参，实现单向的值传送。

【例 9-16】判别一个整数数组中各元素的值，若大于 0 则输出该值，若小于等于 0 则输出 0 值。

（1）在 Visual C++ 6.0 中，新建名称为 9-16.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
void nzp(int v)
{
   if(v>0)
   printf("%d ",v);
   else
   printf("%d ",0);
}
main()
{
   int a[5],i;
   printf("input 5 numbers\n");
   for(i=0;i<5;i++)
   {scanf("%d",&a[i]);
   nzp(a[i]);}
}
```

（3）程序运行结果如图 9-23 所示。

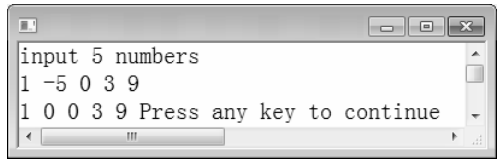

图 9-23　程序运行结果 16

本程序中首先定义一个无返回值函数 nzp()，并说明其形参 v 为整型变量。在函数体中根据 v 值输出相应的结果。在 main()函数中用一个 for 语句输入数组各元素，每输入一个就以该元素作为实参调用一次 nzp()函数，即把 a[i]的值传送给形参 v，供 nzp()函数使用。

9.5.3　数组名作为函数参数

用数组名作函数参数时，要求形参和相对应的实参都必须是类型相同的数组，都必须有明确的数组说明。当形参和实参二者的类型不一致时，即会发生错误。

在用数组名作为函数参数时，不是进行值的传送，即不是把实参数组的每一个元素的值都赋予形参数组的各个元素。因为实际上形参数组并不存在，编译系统不为形参数组分配内存。那么，数据的传送是如何实现的呢？数组名就是数组的首地址。因此，在数组名作为函数参数时所进行的传送只是地址的传送，也就是说把实参数组的首地址赋予形参数组名。形参数组名取得该首地址之后，也就等于有了实在的数组。实际上是形参数组和实参数组为同一数组，共同拥有一段内存空间，如图 9-24 所示。

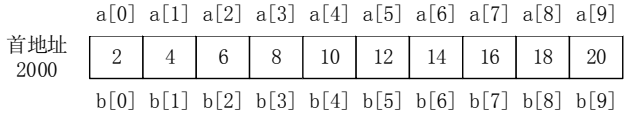

图 9-24　形参数组和实参数组拥有空间情况

设图 9-24 中 a 为实参数组，类型为整型，a 占有以 2000 为首地址的一块内存区；b 为形参数组。当发生函数调用时，进行地址传送，把实参数组 a 的首地址传送给形参数组 b，于是 b 也取得该地址 2000。于是 a、b 两数组共同占有以 2000 为首地址的一段连续内存单元。从图 9-24 中还可以看出 a 和 b 下标相同的元素实际上也占相同的两个内存单元（在编译器 Turbo C 2 中整型数组每个元素占二个字节，在 Visual C++ 6.0 中占四个字节）。

例如，a[0]和 b[0]都占用 2000 和 2001 单元，当然 a[0]等于 b[0]。类似地则有 a[i]等于 b[i]。

【例 9-17】数组 a 中存放一个学生 5 门课程的成绩，求平均成绩。

（1）在 Visual C++ 6.0 中，新建名称为 9-17.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
float aver(float a[5])
{
   int i;
   float av,s=a[0];
   for(i=1;i<5;i++)
   s=s+a[i];
   av=s/5;
```

```
    return av;
}
void main()
{
    float sco[5],av;
    int i;
    printf("\ninput 5 scores:\n");
    for(i=0;i<5;i++)
    scanf("%f",&sco[i]);
    av=aver(sco);
    printf("average score is %5.2f\n",av);
}
```

（3）程序运行结果如图 9-25 所示。

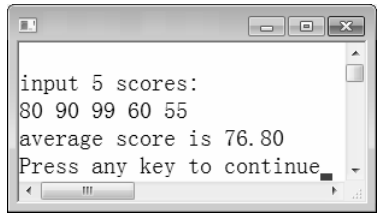

图 9-25　程序运行结果 17

本程序首先定义了一个实型函数 aver()，有一个形参为实型数组 a，长度为 5。在函数 aver()中，把各元素值相加求出平均值，返回给主函数。主函数 main()中首先完成数组 sco()的输入，然后以 sco()作为实参调用 aver()函数，函数返回值传送给 av，最后输出 av 值。从运行情况可以看出，程序实现了所要求的功能。

前面已经研究过，在变量作为函数参数时，所进行的值传送是单向的。即只能从实参传向形参，不能从形参传回实参。形参的初值和实参相同，而形参的值发生改变后，实参并不变化，两者的终值是不同的。而当用数组名作为函数参数时，情况则不同。由于实际上形参和实参为同一数组，因此当形参数组发生变化时，实参数组也随之变化。当然，这种情况不能理解为发生了“双向”的值传递。但从实际情况来看，调用函数之后实参数组的值将因形参数组值的变化而变化。

【例 9-18】题目同例 9-17，改用数组名作为函数参数。

（1）在 Visual C++ 6.0 中，新建名称为 9-18.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
void nzp(int a[5])
{
    int i;
    printf("\nvalues of array a are:\n");
    for(i=0;i<5;i++)
    {
        if(a[i]<0) a[i]=0;
        printf("%d",a[i]);
    }
}
```

```
main()
{
   int b[5],i;
   printf("\ninput 5 numbers:\n");
   for(i=0;i<5;i++)
   scanf("%d",&b[i]);
   printf("initial values of array b are:\n");
   for(i=0;i<5;i++)
   printf("%d",b[i]);
   nzp(b);
   printf("\nlast values of array b are:\n");
   for(i=0;i<5;i++)
   printf("%d",b[i]);
}
```

（3）程序运行结果如图 9-26 所示。

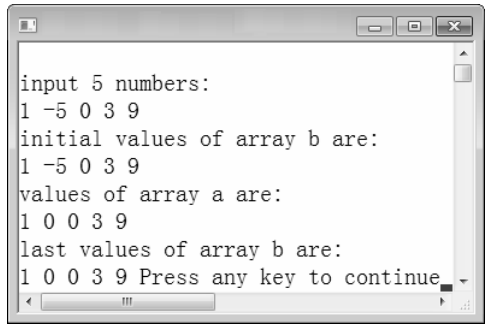

图 9-26　程序运行结果 18

本例中函数 nzp()的形参为整型数组 a，长度为 5。主函数中实参数组 b 也为整型，长度也为 5。在主函数中首先输入数组 b 的值，然后输出数组 b 的初始值。然后以数组 b 为实参调用 nzp()函数。在 nzp()中，按要求把负值单元清 0，并输出形参数组 a 的值。返回主函数之后，再次输出数组 b 的值。从运行结果可以看出，数组 b 的初值和终值是不同的，数组 b 的终值和数组 a 是相同的。这说明实参和形参为同一数组，它们的值同时得以改变。

用数组名作为函数参数时还应注意以下几点。

（1）形参数组和实参数组的类型必须一致，否则将引起错误。

（2）形参数组和实参数组的长度可以不相同，因为在调用时，只传送首地址而不检查形参数组的长度。当形参数组的长度与实参数组不一致时，虽不至于出现语法错误（编译能通过），但程序执行结果将与实际不符，这是应予以注意的。

【例 9-19】编写程序，修改例 9-18。

（1）在 Visual C++ 6.0 中，新建名称为 9-19.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
void nzp(int a[8])
{
   int i;
```

```
    printf("\nvalues of array a are:\n");
    for(i=0;i<8;i++)
    {
        if(a[i]<0)a[i]=0;
        printf("%d",a[i]);
    }
}
main()
{
    int b[5],i;
    printf("\ninput 5 numbers:\n");
    for(i=0;i<5;i++)
        scanf("%d",&b[i]);
    printf("initial values of array b are:\n");
    for(i=0;i<5;i++)
        printf("%d",b[i]);
    nzp(b);
    printf("\nlast values of array b are:\n");
        for(i=0;i<5;i++)
        printf("%d",b[i]);
}
```

（3）程序运行结果如图 9-27 所示。

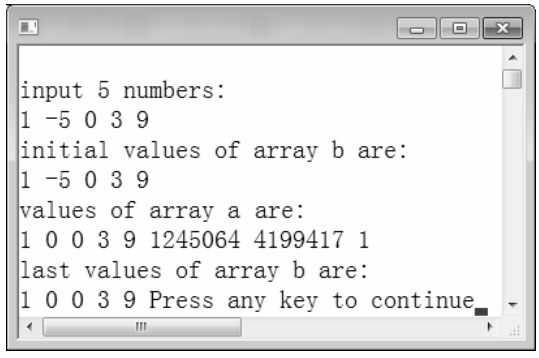

图 9-27　程序运行结果 19

本例与例 9-18 相比，nzp()函数的形参数组长度改为 8，函数体中，for 语句的循环条件也改为 i<8。因此，形参数组 a 和实参数组 b 的长度不一致。编译能够通过，但从结果看，数组 a 的元素 a[5]，a[6]，a[7]显然是无意义的。

在函数形参表中，允许不给出形参数组的长度，或用一个变量来表示数组元素的个数。例如：

```
void nzp(int a[])
```

或者

```
void nzp(int a[],int n)
```

其中，形参数组 a 没有给出长度，而由 n 值动态地表示数组的长度。n 的值由主调函数的实参进行传递。

由此，本例又可改为如下形式。

【例 9-20】形参中数组不确定长度。

（1）在 Visual C++ 6.0 中，新建名称为 9-20.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
void nzp(int a[],int n)
{
    int i;
    printf("\nvalues of array a are:\n");
    for(i=0;i<n;i++)
    {
        if(a[i]<0) a[i]=0;
        printf("%d",a[i]);
    }
}
main()
{
    int b[5],i;
    printf("\ninput 5 numbers:\n");
    for(i=0;i<5;i++)
        scanf("%d",&b[i]);
    printf("initial values of array b are:\n");
    for(i=0;i<5;i++)
        printf("%d",b[i]);
    nzp(b,5);
    printf("\nlast values of array b are:\n");
    for(i=0;i<5;i++)
        printf("%d",b[i]);
}
```

（3）程序运行结果如图 9-28 所示。

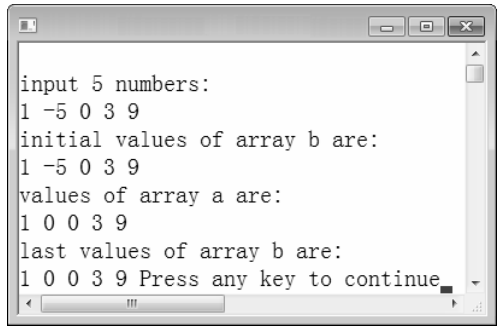

图 9-28　程序运行结果 20

本例中 nzp()函数形参数组 a 没有给出长度，由 n 动态确定该长度。在 main()函数中，函数调用语句为 nzp(b,5)，其中实参 5 将赋予形参 n 作为形参数组的长度。

多维数组也可以作为函数的参数。在函数定义时对形参数组可以指定每一维的长度，也可省去第一维的长度。

例如：

```
int MA(int a[3][10])
```

或者

```
int MA(int a[][10])
```

9.6　内部函数和外部函数

函数一旦定义完成，就可以被其他函数调用。但是实际的开发项目功能模块的划分是相当复杂的，不同的模块将会被写入到不同的源文件中，并由不同的程序员来分别完成。不同的文件之间要共同构成一个有机的程序，那么文件与文件之间也需要交流。因此，多文件中函数的相互调用也必不可少。本节介绍多文件程序中的函数的调用方法。

当一个源程序由多个源文件组成时，C 语言根据函数能否被其他源文件中的函数调用，将函数分为内部函数和外部函数。

9.6.1　内部函数

如果在一个源文件中定义的函数只能被本文件中的函数调用，而不能被同一源程序其他文件中的函数调用，这种函数称为内部函数。在定义内部函数时需要在函数名和函数类型前面加 static 关键字。

内部函数定义的一般形式如下。

```
static 类型说明符 函数名([形参表])
{
    函数体
}
```

其中，[ ]中的部分是可选项，即该函数可以是有参函数，也可以是无参函数。如果为无参函数，形参表为空，但括号必须要有。例如：

```
static int f(int a,int b)                /*内部函数前面加 static 关键字*/
{
   …
}
```

此处，f()函数只能被本文件中的函数调用，在其他文件中不能调用此函数。

内部函数又称静态函数。但此处静态（static）的含义并不是指存储方式，而是指对函数的作用域仅局限于本文件，因此在不同的源文件中定义同名的内部函数不会引起混淆。通常把只由同一个文件使用的函数和外部变量放在一个文件中，前面加上 static 使之局部化，其他文件不能引用。

内部函数的使用会增加函数的访问限制，这增强了程序的健壮性和安全性。特别是在一个具有一定规模的实际项目中，不同的人编写不同的函数时，大家都不必担心自己定义的函数是否会与其他文件中的函数同名。这时，添加一定的限制是非常有必要的。

9.6.2 外部函数

如果在一个源文件中定义的函数可以被同一源程序其他文件中的其他函数调用，这种函数称为外部函数。外部函数在整个源程序中都有效，在定义外部函数时需要在函数名和函数类型前面加关键字 extern。

外部函数定义的一般形式为：

```
extern 类型说明符 函数名(<形参表>)
{
   函数体
}
```

例如：

```
extern int f(int a,int b)                /*外部函数前面加 extern 关键字*/
{
   …
}
```

因为函数与函数之间都是并列的，函数不能嵌套定义，所以函数在本质上都具有外部性质。因此，在定义函数时可以省去 extern 关键字，此时则隐含为外部函数。可以说，前面实例中使用的函数都是外部函数。

如果一个函数定义为外部函数，则这个函数不仅可以被定义它的源文件调用，而且可以被其他的文件中的函数调用，即其作用范围不只局限于本源文件，而是整个程序的所有文件。在一个源文件的函数中调用其他源文件中定义的外部函数时，通常使用 extern 说明被调函数为外部函数。下面就来看一个实例。

【例 9-21】调用外部函数。

（1）在 Visual C++ 6.0 中，新建名称为 9-21.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
/*以下程序调用 mainfile.c 文件*/
#include"stdio.h"
#include"string.h"
int main()
{
   extern void getString(char str[]);
   extern void output(char str[]);
   extern void reverse(char str[], int low, int high);
   char text[50];
   printf("请输入字符串,不要超过 50 个字符:\n");
   getString(text);
   reverse(text, 0, strlen(text)-1);
   printf("反转后的字符串为:\n");
   output(text);
   return 1;
}
/*以下程序调用 input.c 文件*/
#include"stdio.h"
```

```
/*获取字符串*/
void getString(char str[])
{
   gets(str);
}
/*以下程序调用 output.c 文件*/
#include"stdio.h"
/*输出字符串*/
void output(char str[])
{
   printf("%s\n", str);
}
/*以下程序调用 process.c 文件*/
#include"stdio.h"
/*字符串反转处理函数*/
void reverse(char s[], int l, int h)
{
   if(l>h)return;
   else
   {
       char t;
       reverse(s, l+1, h-1);
       t=s[l], s[l]=s[h], s[h]=t;
   }
}
```

假如此程序包含如下 4 个源文件。

```
mainfile.c
output.c
input.c
process.c
```

（3）程序运行结果如图 9-29 所示。

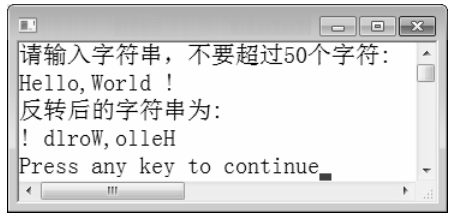

图 9-29　程序运行结果 21

本例的作用是接收一个字符串，然后将该字符串反转，并输出结果。该程序的主函数位于文件 mainfile.c 中，且主函数调用了函数 getString()、output()和 reverse()，而这 3 个函数又分别位于文件 input.c、output.c 和 process.c 中。为了能够让主函数成功地调用它们，我们使用了关键字 extern。

9.7 局部变量和全局变量

在前面的学习中我们知道，形参变量只在被调用期间才分配内存单元，调用结束立即释放。也就是说，形参变量只有在函数内才是有效的，离开该函数就不能再使用了。这种变量有效性的范围称变量的作用域。不只对于形参变量，C 语言中所有的量都有自己的作用域。变量说明的方式不同，其作用域也不同。C 语言中的变量，按作用域范围可分为两种，即局部变量和全局变量。

9.7.1 局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内，离开该函数后再使用这种变量是非法的。

例如：

```
int f1(int a)        /*函数 f1()*/
{
   int b,c;
   …
}
a,b,c 有效
int f2(int x)        /*函数 f2()*/
{
   int y,z;
   …
}
x,y,z 有效
main()
{
   int m,n;
   …
}
m,n 有效
```

在函数 f1()内定义了三个变量，a 为形参，b,c 为一般变量。在 f1()的范围内 a,b,c 有效，或者说 a,b,c 变量的作用域限于 f1()内。同理，x,y,z 的作用域限于 f2()内。m,n 的作用域限于 main()函数内。关于局部变量的作用域还要说明以下几点：

（1）主函数中定义的变量也只能在主函数中使用，不能在其他函数中使用。同时，主函数中也不能使用其他函数中定义的变量。因为主函数也是一个函数，它与其他函数是平行关系。这一点是与其他语言不同的，应予以注意。

（2）形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。

（3）允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰，也不会发生混淆。

（4）在复合语句中也可定义变量，其作用域只在复合语句范围内。

例如：

```
main()
{
    int s,a;
    …
    {
        int b;
        s=a+b;
        …                    /*b 作用域*/
    }
    …                        /*s,a 作用域*/
}
```

【例 9-22】变量的作用域。

（1）在 Visual C++ 6.0 中，新建名称为 9-22.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
main()
{
    int i=2,j=3,k;
    k=i+j;
    {
        int k=8;
        printf("%d\n",k);
    }
    printf("%d\n",k);
}
```

（3）程序运行结果如图 9-30 所示。

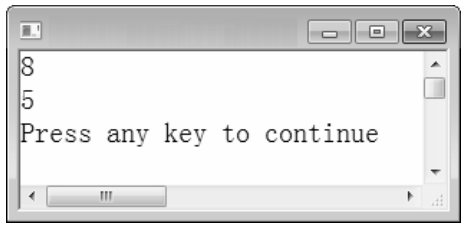

图 9-30　程序运行结果 22

本例在 main()中定义了 i，j，k 三个变量，其中 k 未赋初值。而在复合语句内又定义了一个变量 k，并赋初值为 8（注意这两个 k 不是同一个变量）。在复合语句外由 main()定义的 k 起作用，而在复合语句内则由在复合语句内定义的 k 起作用。因此程序第 4 行的 k 为 main()所定义，其值应为 5。第 7 行输出 k 值，该行在复合语句内，由复合语句内定义的 k 起作用，其初值为 8，故输出值为 8，第 9 行输出 i，k 值。i 是在整个程序中有效的，第 7 行对 i 赋值为 3，故输出也为 3。而第 9 行已在复合语句之外，输出的 k 应为 main()所定义的 k，此 k 值由第 4 行已获得为 5，故输出也为 5。

9.7.2 全局变量

所谓全局变量，是指在函数外部定义的变量，是相对于局部变量而言的。全局变量又称外部变量。全局变量不属于哪一个函数，它属于一个源程序文件。其作用域是整个源程序，也就是说它的有效范围是从该变量定义的位置处开始直到源文件的结尾。

在函数中使用全局变量，一般应作全局变量声明。只有在函数内经过声明的全局变量才能使用。全局变量的声明要用关键字 extern。但在一个函数之前定义的全局变量，在该函数内无须声明即可使用。

例如：

```
int a,b;                              /*外部变量()*/
void f1()                             /*函数 f1()*/
{
   …
}
float x,y;                            /*外部变量*/
int f2()                              /*函数 f2()*/
{
   …
}
main()                                /*主函数*/
{
   …
}
double m,n;                           /*全局变量 x、y、a、b、m、n 的作用域*/
```

从上例可以看出，a、b、x、y、m、n 都是在函数外部定义的外部变量，都是全局变量。但 x、y 定义在函数 f1()之后，而在 f1()内又无对 x、y 的声明，所以它们在 f1()内无效。a、b 定义在源程序最前面，因此在 f1()、f2()及 main()内不加说明也可使用。m、n 定义在所有函数之后，因此不能被任何函数使用。

使用全局变量的一个最主要作用就是允许多个函数都对某个变量进行修改，这就意味着全局变量保持了一种方便多个函数有效沟通的渠道。

【例 9-23】输入长方体的长、宽、高，求体积及 3 个面的面积。

（1）在 Visual C++ 6.0 中，新建名称为 9-23.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include"stdio.h"
int s1,s2,s3;                    /*全局变量 s1、s2、s3*/
int vs(int length,int width,int height)
{
   int v;
   v=length*width*height;
   s1=length*width;
   s2=width*height;
   s3=length*height;
   return v;
}
```

```
main()
{
   int v,l,w,h;
   printf("input length,width and height:\n");
   scanf("%d%d%d",&l,&w,&h);
   v=vs(l,w,h);
   printf("v=%d s1=%d s2=%d s3=%d\n",v,s1,s2,s3);
}
```

（3）程序运行结果如图 9-31 所示。

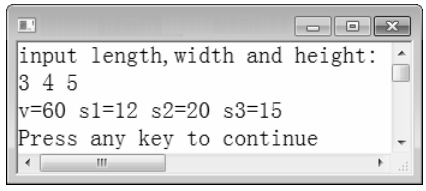

图 9-31　程序运行结果 23

本例中定义了 3 个外部变量 s1、s2、s3， 用来存放 3 个面积，其作用域为整个程序。函数 vs 用来求正方体体积和 3 个面积，函数的返回值为体积 v。由主函数完成长、宽、高的输入及结果输出。由于 C 语言规定函数返回值只有一个，当需要增加函数的返回数据时，用外部变量是一种很好的方式。此例中，如不使用外部变量，在主函数中就不可能取得 v、s1、s2、s3 这 4 个值。而采用了外部变量，在函数 vs 中求得的 s1、s2、s3 值在 main()中仍然有效。因此，外部变量是实现函数之间数据通信的有效手段。

通过例 9-23 发现，如果需要传递多个数据，除了使用函数值外，还可以借助于全局变量，因为函数的调用只能带回一个返回值，因此有时可以利用全局变量增加与函数联系的渠道，从函数得到多个返回值。因此，全局变量的使用增加了函数之间传送数据的途径。在全局变量的作用域内，任何一个函数都可以引用该全局变量。但如果在一个函数中改变了全局变量的值，就能影响到其他函数，相当于各个函数间有直接的传递通道。

注意：如果在该函数内定义了一个与之前定义的全局变量同名的变量，那么该同名局部变量就会在函数内部屏蔽全局变量的影响。

下面来看一个内外变量同名的实例。

【例 9-24】全局变量和局部变量同名的实例，输出两个数中的较大者。

（1）在 Visual C++ 6.0 中，新建名称为 9-24.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
int a=3,b=5;                    /*全局变量 a、b*/
int max(int a,int b)            /*局部变量 a、b*/
{
   int c;
   c=a>b?a:b;
   return c;
}
main()
```

```
{
    int a=8;                        /*局部变量 a*/
    printf("%d\n",max(a,b));
}
```

（3）程序运行结果如图 9-32 所示。

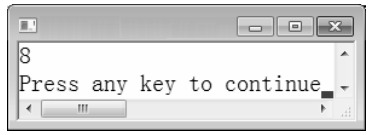

图 9-32　程序运行结果 24

本例中定义了两个全局变量 a 和 b，在 main()函数中定义了局部变量 a，根据局部变量优先的原则，main()函数中函数调用的实参 a 是 8，b 的值是全局变量 5，因此程序的运行结果比较的是 8 和 5 的较大值。

当局部变量与全局变量同名时，在局部变量的作用范围内，全局变量不起作用，即局部变量优先。

为了便于区别全局变量和局部变量，在 C 语言中有一个不成文的约定，将全局变量名的第一个字母用大写表示。

尽管全局变量能够方便多函数之间的信息传递，但是仍然建议在非必要时尽量不要使用全局变量。原因有以下几点。

（1）外部变量破坏了函数的独立性，函数不再以独立的形式来完成各自的功能，因为这些功能的实现将受制于外部因素，这与结构化程序设计的思想是相违背的。

（2）全局变量在程序执行的过程中始终占用存储单元，而不像局部变量那样仅在使用时才被分配存储单元，外部变量的释放必须等到整个程序完结，这也是一个不太经济的做法，因此在不必要时尽量不要使用全局变量。

（3）使用全局变量过多，会降低程序的清晰性，人们往往难以清楚地判断出每个瞬时各个外部变量的值。在各个函数执行时，都可能改变外部变量的值，程序容易出错。因此，要限制使用全局变量，而多使用局部变量。

9.8　综合案例——求方程的根

本节通过一个综合应用的例子，巩固前面学习的函数的定义、函数的调用和参数传递等知识。

【例 9-25】求方程 $x^3-5x^2+16x-80=0$ 在区间[-3,6]内的根。

（1）在 Visual C++ 6.0 中，新建名称为 9-25.c 的 Text File 文件。

（2）在代码编辑区域输入以下代码。

```
#include<stdio.h>
#include<math.h>                    /*下面程序中使用了 pow()等函数,需要包含头文件 math.h*/
float func(float x)                 /*定义 func()函数,用来求函数 func(x)=x*x*x-5*x*x+16x-80 的值*/
{
    float y;
    y=pow(x,3)-5*x*x+16*x-80.0f;    /*计算指定 x 值的 func(x)的值,赋给 y*/
```

```
    return y;                          /*返回 y 的值*/
}
/*定义 point_x()函数,用来求出弦在[x1,x2]区间内与 X 轴的交点*/
float point_x(float x1,float x2)
{
    float y;
    y=(x1*func(x2)-x2*func(x1))      /(func(x2)-func(x1));
    return y;
}
float root(float x1,float x2)        /*定义 root()函数,计算方程的近似根*/
{
    float x,y,y1;
    y1=func(x1);                       /*计算 x 值为 x1 时的 func(x1)函数值*/
    do{                                /*循环执行下面的语句*/
        x=point_x(x1,x2);              /*计算连接 func(x1)和 func(x2)两点的弦与 X 轴的交点*/
        y=func(x);                     /*计算 x 点对应的函数值*/
        if(y*y1>0)                     /*func(x)与 func(x1)同号,说明根在区间[x,x2]内*/
        {
             y1=y;                     /*将此时的 y 作为新的 y*/
             x1=x;                     /*将此时的 x 作为新的 y*/
        }
        Else                           /*否则将此时的 x 作为新的 x*/
        {
             x2=x;
        }
    }while(fabs(y)>=0.0001);
    return x;                          /*返回根 x 的值*/
}
void main()
{
    float x1=-3,x2=6;
    float t=root(x1,x2);
    printf("方程的根为: %f\n",t);
}
```

（3）程序运行结果如图 9-33 所示。

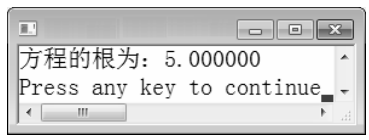

图 9-33　程序运行结果 25

本例是用截弦法求方程的根，方法如下。

（1）取两个不同的点 x1 和 x2，如果 f(x1)、f(x2)符号相反，则区间(x1,x2)内必有一个根；如果 f(x1)、f(x2)符号相同，改变 x1 和 x2 的值直到上述条件成立为止。

（2）连接 f(x1)、f(x2)两点，这个弦就交 X 轴于 x 处，那么 x 点的坐标就可以用下面的公式求解：

x=(x1*func(x2)−x2*func(x1))/(func(x2)−func(x1))，由此可以进一步求出 x 点对应的 f(x)。

（3）如果 f(x)、f(x1)同号，则根必定在(x,x2)区间内，此时将 x 作为新的 x1。如果 f(x)、f(x1)异号，表示根在(x1,x)区间内，此时可将 x 作为新的 x2。

（4）重复步骤（1）、（2），直到|f(x)|<ε 为止，ε 为一个很小的数，程序中设为 0.0001，此时可认为 f(x)≈0。

9.9 就业面试技巧与解析

本章是整个 C 语言的核心内容之一，本章讲述的函数是 C 语言的最小组成部分。在各种面试中，本章知识都深得 HR 喜爱，经常会提问，因此必须要牢固掌握。

9.9.1 面试技巧与解析（一）

本章的知识主要是函数，也有变量的作用域以及存储类型等相关知识，都非常重要，在学习的时候务必多理解、多练习，因为这些即是编好程序的基础知识，又是赢得面试的重要砝码。

9.9.2 面试技巧与解析（二）

面试官：局部变量能否和全局变量重名？

应聘者：能。如果重名，局部变量会屏蔽全局变量。

要用全局变量，需要使用“::”，局部变量可以与全局变量同名，在函数内引用这个变量时，会用到同名的局部变量，而不会用到全局变量。对于有些编译器而言，在同一个函数内可以定义多个同名的局部变量，如在两个循环体内都定义一个同名的局部变量，而那个局部变量的作用域就在那个循环体内。

面试官：如何引用一个已经定义过的全局变量？

应聘者：extern。可以用引用头文件的方式，也可以用 extern 关键字。如果用引用头文件方式来引用某个在头文件中声明的全局变量，假定将那个变量写错了，那么在编译期间会报错，如果用 extern 方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间会报错。

面试官：全局变量可不可以定义在可被多个，C 文件包含的头文件中？为什么？

应聘者：可以，在不同的 C 文件中以 static 形式来声明同名全局变量。可以在不同的 C 文件中声明同名的全局变量，前提是其中只能有一个 C 文件中对此变量赋初值，此时连接不会出错。

面试官：关键字 static 的作用是什么？

应聘者：这个简单的问题很少有人能回答完全。在 C 语言中，关键字 static 有三个明显的作用。

（1）在函数体，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。

（2）在模块内（在函数体外），一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其他函数访问。它是一个本地的全局变量。

（3）在模块内，一个被声明为静态的函数只可被这一模块内的其他函数调用。也就是说，这个函数被限制在声明它的模块的本地范围内使用。