第 8 章　函数

本章首先总括地说明 C 语言的函数与程序结构，然后介绍函数的声明、函数的定义、函数的参数、函数的返回值、函数的调用、函数的嵌套调用和函数的递归调用等内容，介绍了以数组元素和数组名作为函数的参数的程序设计，还介绍了局部变量与全局变量以及变量的存储类别。通过一个具体的多源文件程序的例子说明了工程文件的创建和生成可执行文件的方法。最后介绍了编译预处理的有关知识。

- **8.1** 概述
- **8.2** 函数的语法
- **8.3** 数组作为函数参数
- **8.4** 函数的嵌套调用
- **8.5** 函数的递归调用
- **8.6** 局部变量和全局变量
- **8.7** 变量的存储类别
- **8.8** 工程文件
- **8.9** 编译预处理

8.1　概述

前面，已经介绍了三种基本结构的程序设计，这一章里，将介绍能够实现模块化程序设计的一种程序形式——子程序调用。在其他许多高级语言中都有子程序这个概念，所谓子程序就是将经常需要重复使用的程序编写成为一个相对独立的通用程序，当需要使用该通用程序一次时，就调用该程序一次，这个通用的程序，就称为子程序。而调用子程序的程序则称为主程序。主程序调用子程序的过程可以通过图 8.1 来说明。

在 C 语言中，子程序的作用是由函数完成的，调用子程序，就是调用函数。使用主函数调用另一个函数的示意图如图 8.2 所示。可以看出，当主函数调用另一个函数时，程序将自动转到被调用函数去执行该函数，当该函数执行完毕，程序自动返回主函数继续执行。这样的程序形式与前面介绍的三种基本结构的程序是不同的。利用函数的调用，一方面，将一些常用的功能模块编写成函数，放在函数库中供程序调用，从而减少程序段的重复编写；另一

方面，将较大的程序分为若干个程序模块，每个模块用来实现一个特定的功能，从而实现模块化程序设计。

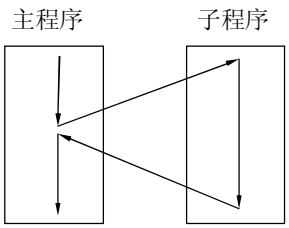


图 8.1　主程序调用子程序示意图

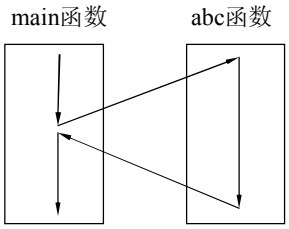


图 8.2　主函数 main()调用另一个函数 abc()示意图

例如，计算组合 $C_n^m = \frac{n!}{m!(n-m)!}$，在算式中多次求阶乘，如果顺序地设计求 n 的阶乘、m 的阶乘和 n-m 的阶乘的程序，则程序变得冗长，并且需要重复书写代码。这时，可以将求阶乘的代码设计成自定义函数，每当需要求阶乘时就调用一次函数，简化了程序，同时也提高了代码的重用性。

实际上，函数的基本概念在使用标准库函数的时候就已经接触到了。例如当在设计程序中需要做求平方根的计算时，就会使用到库函数中的数学函数 sqrt()函数，这就是函数的调用，只是这个函数是编译系统提供的，而自定义函数则是程序员按照自己的需要来设计的。

一个 C 语言程序可由一个主函数和若干个函数构成。在 C 语言中，主函数可以调用其他函数，其他函数之间也可以互相调用。同一个函数可以被一个或多个函数调用任意多次。为了叙述方便，将调用自定义函数的函数称为主调函数，而将被调用的自定义函数称为被调函数。下面，通过一个简单的函数调用例子，来说明函数调用程序的执行过程，从而建立起函数调用的概念。

例 8.1　函数 printstar()的功能为打印一行*号，函数 print_message()的功能为打印“Happy New Year!”。设计程序，通过调用这两个函数，按下列格式在屏幕上输出图案和文字：

```
**********
    Happy New Year!
**********
```

程序：

```
#include "stdio.h"
main()
{
 void printstar();                    /*声明 printstar 函数*/
 void print_message();                /*声明 print_message 函数*/
 printstar();                         /*调用 printstar 函数*/
 print_message();                     /*调用 print_message 函数*/
 printstar();                         /*调用 printstar 函数*/
}
void printstar()                      /*printstar 函数*/
{
 printf("**********\n");
}
void print_message()                  /*print_message 函数*/
{
```

```
 printf("  Happy New Year!\n");
}
```

运行：

```
**********************
   Happy New Year!
**********************
```

在以上程序中，除了 main()函数外，还编写了两个函数，一个是 printstar()函数，另一个是 print_message()函数。printstar()函数完成显示一行星号的功能，print_message()函数则完成显示字符串“Happy New Year!”的功能。主函数由五个语句组成，第一个语句是对 printstar()函数进行声明；第二个语句是对 print_message()函数进行声明；第三个语句是 printstar()函数调用语句，执行该函数调用后，在屏幕上显示一行星号；然后执行第四个语句，第四个语句是 print_message()函数调用语句，执行该函数调用后，在屏幕上显示“Happy New Year!”；最后执行第五个语句，第五个语句是 printstar()函数调用语句，执行该函数调用后，又显示一行星号。

在 C 语言中，一个 C 语言源程序文件可由一个或多个函数组成。一个 C 语言源程序文件是一个编译单位，即以源程序为单位进行编译，一个源程序文件经过编译产生一个目标文件。C 语言中，一个 C 语言程序还可以由一个或多个源程序文件组成。对于较大的程序，一般是将若干个函数以及其他内容分别放在若干个源程序文件中，再由若干个源程序文件组成一个 C 语言程序。这样可以由各个项目组的人员分别编写、分别编译，从而提高工作效率。一个源程序文件可以为多个 C 语言程序公用。main 函数是系统定义的。C 语言程序的执行从 main 函数开始，调用其他函数后流程返回到 main 函数，在 main 函数中结束整个程序的运行。在程序中，所有函数都是平行的，即在定义函数时是互相独立的，一个函数不从属于另一函数，即函数不能嵌套定义。函数间可以互相调用，但是 main 函数不能被其他函数调用。

从用户使用的角度看，函数可分为两类：一类是标准库函数，标准库函数是由系统定义的，用户可以直接使用，但在使用前要将相应的头文件包含到程序中去。另一类是用户自己定义的函数，是用户根据需要自己定义的函数，称为自定义函数。自定义函数用以解决用户专门需要解决的问题，为用户提供了极大的方便，同时还提高了程序代码的重用性。

从函数的形式看，函数可分为两类：一类是无参函数，另一类是有参函数。在函数名后面的圆括号内没有参数的函数称为无参函数，例如，例 8.1 中的 printstar 和 print_message 函数就是无参函数。而在函数名后面的圆括号内有参数的函数则称为有参函数。有参函数在调用函数时，在主调函数和被调函数之间有数据的传递。也就是说，主调函数可以将数据传给被调函数使用，被调用函数中的数据也可以带回来供主调函数使用。例如下面将要讲到的求最大值的函数 max 就是一个有参函数。

8.2　函数的语法

8.2.1　函数定义的一般格式

1. 无参函数的定义形式

无参函数的定义形式为：

类型标识符 函数名()

{声明部分

语句

}

例 8.1 中的 printstar()和 print_message()函数就是无参函数，其代码如下所示。

```
void printstar()                    /*printstar 函数*/
{
 printf("********************\n");
}
void print_message()                /*print_message 函数*/
{
 printf("  Happy New Year!\n");
}
```

以 printstar()函数为例，其中的 void 是类型标识符，**类型标识符**表示了函数返回值的类型，可以是 int、float、double、char 等，这里的 void 则表示该函数无返回值。printstar 是函数名，在定义一个自定义函数时，必须给函数命名，**函数名**用标识符表示。由于该程序中没有使用变量，因此省略了**声明部分**，如果在函数中使用了变量，则必须在**声明部分**对所使用的变量进行定义。该函数的**语句**部分则是一个 printf 函数调用语句：

```
printf("********************\n");
```

函数定义形式中的**语句**部分可以是若干个语句。

2. 有参函数定义的一般形式

有参函数定义的一般形式为：

类型标识符 函数名（形式参数列表）

{声明部分

语句

}

有参函数与无参函数的定义类似，区别在于有参函数的函数名后的圆括号中有形式参数的列表。形式参数的列表的一般形式为：

类型名 $_1$ 参数 $_1$，类型名 $_2$ 参数 $_2$，……，类型名 $_n$ 参数 $_n$

例 8.2 求出两个任意整数中的最大数，并输出。

程序：

```
#include "stdio.h"
main()
{int a,b,c;
 int max(int x,int y);              /*声明有参函数 max*/
 printf("请输入 a,b:");
 scanf("%d,%d",&a,&b);
 c=max(a,b);
 printf("max=%d\n",c);
}
int max(int x,int y)                /*定义有参函数 max*/
{int z;                             /*函数中的说明部分*/
 if (x>y)
   z=x;
 else
   z=y;
 return(z);
}
```

运行：

```
2，8↓
max=8
```

3. 空函数

C 语言中，函数的构成部分是可以省略的，函数最简单的形式是空函数，空函数的形式如下所示：

类型标识符 函数名()

{}

例如最简单的形式可以为：

abc()

{ }

当省略了**类型标识符**时，则默认为 int 类型。调用此函数时，程序进入 abc 函数后没有执行任何语句就返回了主调函数。从表面上看程序没有做任何操作，但实际上，空函数在程序开发中是很有用处的。在实际设计程序时，往往将系统设计为若干个模块，分到各个项目小组去设计，在开发期间，可以使用空函数来为模块保留位置，留待以后来设计代码。

例 8.3　空函数应用举例，学生成绩管理系统的模块示意图如图 8.3 所示。编写一个学生成绩管理系统菜单程序，其中输入、修改、统计、查询和打印 5 个模块所对应的函数 z1()、z2()、z3()、z4()和 z5()留待以后填充代码。

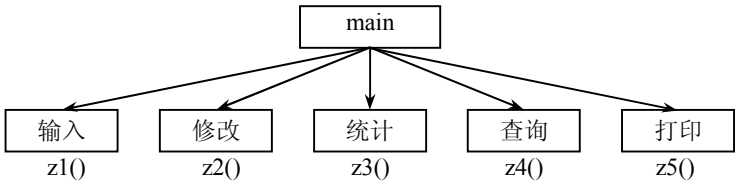


图 8.3　学生成绩管理系统模块

程序：

```
#include "stdio.h"
main()
{
  int i,m=1;
  while (m!=6)
  {
   clrscr();                          /*清屏*/
   printf("1.输入\n");                /*显示菜单*/
   printf("2.修改\n");
   printf("3.统计\n");
   printf("4.查询\n");
   printf("5.打印\n");
   printf("6.退出\n");
   do                                 /*容错*/
   {
    printf("请按需输入(1~6):\n");
    scanf("%d",&m);
   } while (m>6 || m<1);
   if (m==1)
    z1();
   else if (m==2)
```

```
      z2();
    else if (m==3)
      z3();
    else if (m==4)
      z4();
    else if (m==5)
      z5();
   }
}
z1()                    /*输入模块，尚待开发*/
{ }
z2()                    /*修改模块，尚待开发*/
{ }
z3()                    /*统计模块，尚待开发*/
{ }
z4()                    /*查询模块，尚待开发*/
{ }
z5()                    /*打印模块，尚待开发*/
{ }
```

4. 对形参声明的传统方式

在老版本的 C 语言中，对形式参数类型的声明是放在函数定义的第二行，也就是不在第一行的括号内指定形式参数的类型，而在括号外单独指定，例如上面定义的 max 函数可以写成以下形式：

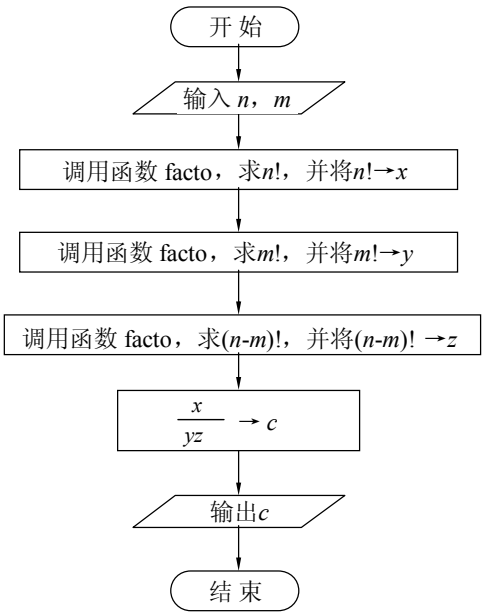
```
/*求两个任意整数中的最大值*/
/*传统形式参数说明方式*/
#include "stdio.h"
main()
{int a,b,c;
 int max(int x,int y);
 printf("Please input a,b:");
 scanf("%d,%d",&a,&b);
 c=max(a,b);
 printf("max=%d\n",c);
}
int max(x,y)                    /*指定参数 x,y*/
int x,y;                        /*对参数进行类型说明*/
{int z;
 if (x>y)
   z=x;
 else
   z=y;
 return(z);
}
```

一般把这种方法称为传统的对形式参数的声明方式，而把前面介绍的方法称为现代的方式。Turbo C 和目前使用的多数 C 语言版本对这两种方法都允许使用，两种方法等价，ANSI 新标准推荐使用前一种方法，即现代方式。

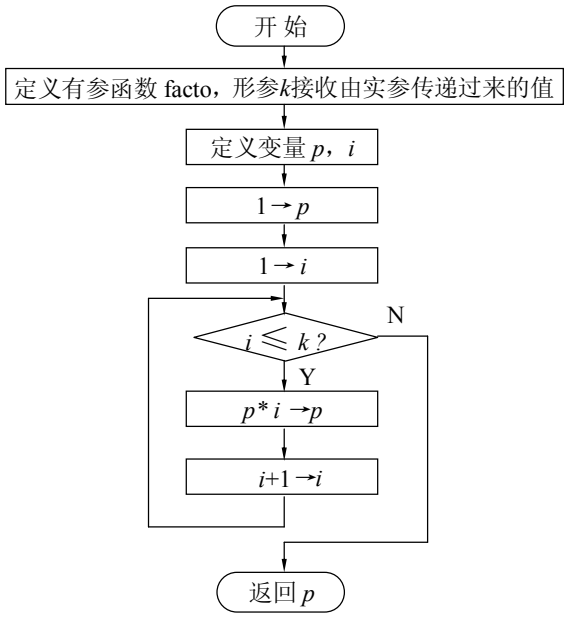
下面举一个函数应用的例子，通过这个例子来了解有关函数的程序设计基本概念。

例 8.4 计算组合 $C_n^m=\dfrac{n!}{m!(n-m)!}$，其中 n 和 m 通过键盘输入。

在程序中，设计一个自定义函数 facto，facto 函数的作用为计算一个正整数的阶乘。程序的流程图如图 8.4 所示，其中 main 函数的流程图如图 8.4（a）所示，facto 函数的流程图如图 8.4（b）所示，程序的源代码如下所示：



（a）main 函数的流程图



（b）facto 函数的流程图

图 8.4　计算组合流程图

```
#include "stdio.h"
main()
{long int facto(long int k);          /*函数原型*/
 long int m,n,x,y,z,c;
 printf("n=");
 scanf("%ld",&n);
 printf("m=");
 scanf("%ld",&m);
 x=facto(n);
 y=facto(m);
 z=facto(n-m);
 c=x/(y*z);
 printf("c=%ld\n",c);
}
 long int facto(long int k)           /*定义有参函数 facto,功能为求 k!*/
{
 long int p,i;                        /*函数中的说明部分*/
 p=1;
 for(i=1;i<=k;i++)
   p=p*i;
 return(p);
}
```

运行：

```
n=10↓
m=6↓
c=210
```

在以上的程序中包含了函数的定义、函数的声明、函数的形参与实参、函数的调用以及函数值的返回等有关函数的基本知识，下面的小节将一一对这些函数的基本概念做详细的讲解，具体介绍函数的语法。

8.2.2 自定义函数的声明

如果使用用户自定义函数，而且该函数与调用它的函数在同一个文件中，一般还应在主调函数中对被调用的函数做声明。所谓声明是指向编译系统声明将要调用的函数，并将有关信息通知编译系统。对函数声明称为函数原型。函数原型的作用主要是利用它在程序的编译阶段对函数的合法性做全面的检查。函数原型的一般形式如下所示。

形式 1：

函数类型 函数名（参数类型 $_1$，参数类型 $_2$，……）；

形式 2：

函数类型 函数名（参数类型 $_1$ 参数名 $_1$，参数类型 $_2$ 参数名 $_2$，……）；

这里对有关声明函数做一些说明，如果在函数调用之前没有对函数做声明，则编译系统把第一次遇到的函数形式比如函数定义或者函数调用作为函数声明，并将函数类型默认为 int 类型。如果被调用函数的定义出现在主调用函数之前，可以不必加以声明。因为编译系统已经先知道了定义的函数类型，会根据函数首部提供的信息对函数的调用做正确性的检查。如果已在所有函数定义之前，在函数的外部已经做了函数声明，则在各个函数中不必对所调用的函数再做声明。

例 8.2 和例 8.4 中的语句

```
int max(int x,int y);            /*例 8.2 程序对函数的声明语句*/
long int facto(long int k);      /*例 8.4 程序对函数的声明语句*/
```

就是对函数的声明，函数的声明也可以放在函数的外部，例如可以将例 8.4 的程序设计成如下形式：

```
#include "stdio.h"
long int facto(long int k);      /*函数原型*/
main()
{
 long int m,n,x,y,z,c;
 printf("n=");
 scanf("%ld",&n);
 printf("m=");
 scanf("%ld",&m);
 x=facto(n);
 y=facto(m);
 z=facto(n-m);
 c=x/(y*z);
 printf("c=%ld\n",c);
}
 long int facto(long int k)      /*定义有参函数 facto,功能为求 k!*/
{
 long int p,i;                   /*函数中的说明部分*/
 p=1;
 for(i=1;i<=k;i++)
   p=p*i;
 return(p);
}
```

8.2.3　函数参数和函数的值

1. 形式参数和实际参数

在调用函数时，大多数情况下，主调函数和被调用函数之间有数据传递关系。这就要用到有参函数。

在定义函数时函数名后面圆括号中的变量名称为形式参数，简称为形参。在主调函数中调用一个函数时，函数名后面圆括号中的参数称为实际参数，简称为实参。

下面通过前面所举的例 8.2 来说明调用函数时的数据传递，例 8.2 的程序代码如下所示：

```
#include "stdio.h"
main()
{int a,b,c;
 int max(int x,int y);                    /*声明有参函数 max*/
 printf("请输入 a,b:");
 scanf("%d,%d",&a,&b);
 c=max(a,b);
 printf("max=%d\n",c);
}
int max(int x,int y)                      /*定义有参函数 max*/
{int z;                                   /*函数中的说明部分*/
 if (x>y)
   z=x;
 else
   z=y;
 return(z);
}
```

运行：

```
2，8↓
max=8
```

在定义函数中指定的形参，在未出现函数调用时，形参并不占内存中的存储单元。只有在发生函数调用时，函数中的形参才由系统分配内存单元。在调用结束后，形参所占的内存单元也被释放。程序中参数传递的示意图如图 8.5 所示。

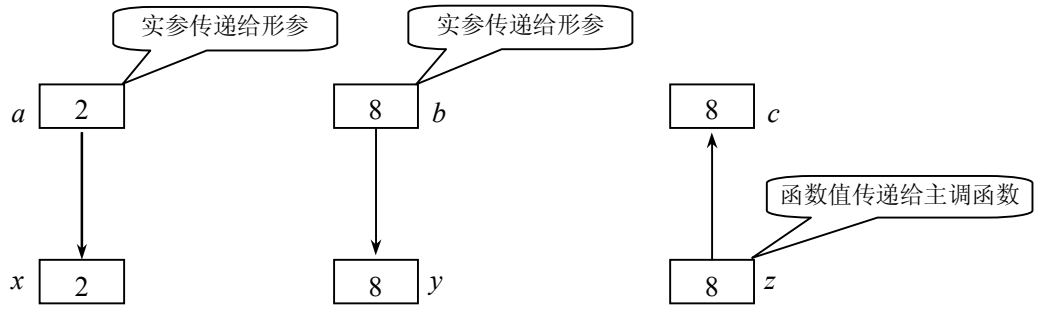


图 8.5　例 8.4 中程序参数传递的示意图

实参可以是常量、变量或表达式。例如：c=max(6,a+b);，但要求它们在函数调用时有确定的值，在调用时将实参的值赋给形参。实参也可以是数组名，如果是数组名，则传递的是数组首地址而不是数组的值。

在 C 语言中参数传递的方式可以分为值传递方式和地址传递方式。当形参为一个简单变

量时，参数是以值传递的方式进行传递的；当形参为一个数组或者一个指针变量时，参数是以地址传递的方式进行传递的。值传递属于单向传递，如图 8.5 所示。单向传递是指只由实参传给形参，而不能由形参传回来给实参。

在设计自定义函数的程序时还应注意实际参数与形式参数的类型应相同或赋值兼容。如果实际参数为整型而形式参数为实型，或者相反，则按照前面介绍的不同类型数据的赋值规则进行转换。例如，实际参数 *a* 的值为 3.5，而形式参数 *x* 为整型，则将实数 3.5 转换成整型数 3，然后送到形式参数 *x*。但此时应将 max 函数放在 main 函数的前面或在 main 函数中对被调用函数 max 做原型声明，否则会出错。由此可见，写出函数原型即对函数进行声明是非常重要的。

例 8.5 分析下列程序是否能够完成交换两个变量的值的操作。

```
/*注意：C 语言中，由于参数传递是单向的，所以这是一个错误的程序*/
#include "stdio.h"
main()
{int a,b;
 a=5;b=10;
 swap(a,b);
 printf("a=%d,b=%d\n",a,b);
}
swap(int x,int y)
{int temp;
 temp=x;
 x=y;
 y=temp;
}
```

运行：

```
a=5,b=10
```

需要注意的是，由于在程序中参数传递方式属于值传递，数据的传递是单向的，因此这个程序不能完成交换两个变量值的操作。

2. 函数的返回值

通常，希望通过函数调用，使主调函数能得到一个确定的值，这就是函数的返回值。函数的返回值是通过 return 语句获得的。

return 语句的一般形式为：

return 表达式;

或

return （表达式）;

功能：计算表达式的值，将表达式的值作为函数的返回值返回到主调函数。

函数可以将多个实参一一传递给多个形参。然而，函数只能有一个返回值。当函数不需要返回值时，return 后面的**表达式**可以省略，简化为：

return;

这时，也可以不写 return 语句，当不写出 return 语句时，函数运行到右花括号时就结束函数的执行，程序返回到主调函数。C 语言还允许在一个函数内书写多个 return 语句，在这种情况下，只要执行到一个 return 语句，都结束函数的执行，程序返回。当函数没有返回值时，应将该函数的类型规定为 void 类型，void 类型称为无值类型或空类型，它表示函数没有返回值。一般函数值的类型和返回值的类型是一致的，当函数值的类型和 return 语句中表达式的值不一致时，则以函数类型为准。计算机对数值型数据可以自动进行类型转换。即函数类型决定返回

值的类型。

例如可以将例 8.2 中的 max 函数书写为：

```
int max(int x,int y)          /*定义有参函数 max*/
{
 if (x>y)
   return(x);
 else
     return(y);
}
```

8.2.4　函数的调用

1. 函数调用的一般形式

函数调用的一般形式为：

函数名（实际参数列表）；

说明：

（1）如果是调用无参函数，则**实际参数列表**可以省略，但是圆括号不能省。

（2）如果实际参数列表中包括多个实际参数，则各个参数之间用逗号隔开。

（3）实际参数与形式参数的个数应相等，类型应一致。实际参数与形式参数按顺序对应，一一传递。

（4）如果实际参数列表中包括多个参数，对实际参数求值的顺序取决于编译系统。有的系统按自左至右，有的系统则按自右至左的顺序。例如 Turbo C 和 MS C 是按自右至左的顺序求值的。

例 8.6　分析下列程序的运行结果。

程序：

```
#include "stdio.h"
main()
{
 int f(int a,int b);
 int i=2,p;
 p=f(i,++i);              /*函数调用*/
 printf("%d",p);
}
int f(int a,int b)
{int c;
 if (a>b) c=1;
 else if(a==b) c=0;
 else c=-1;
 return(c);
}
```

运行：

```
0
```

运行结果分析：因为 Turbo C 是按自右至左的顺序求值的，所以传递的参数为 f(3,3)。

2. 函数调用的方式

常用的函数调用方式有函数调用语句、函数作为表达式的运算对象以及函数作为函数的参数等。

（1）函数调用语句。在函数后面加上分号形成的语句称为函数调用语句。从例 8.1 中的

程序代码可以看到函数调用语句。其部分代码如下所示：

```
main()
{
 void printstar();                  /*声明 printstar 函数*/
 void print_message();              /*声明 print_message 函数*/
 printstar();                       /*调用 printstar 函数*/
 print_message();                   /*调用 print_message 函数*/
 printstar();                       /*调用 printstar 函数*/
}
```

程序中的语句：

```
print_message();                    /*调用 print_message 函数*/
printstar();                        /*调用 printstar 函数*/
```

都属于函数调用语句。

（2）函数出现在表达式中。函数可以作为运算对象出现在表达式中。函数出现在一个表达式中时，要求函数带回一个确定的函数值参加表达式的运算。例如，例 8.4 中的程序可以设计为：

```
c= facto(n)/(facto(m)*facto(n-m));
```

在这个赋值语句中，函数 facto 出现在算术表达式中，做乘法和除法。

（3）函数作为函数参数。函数也可以作为一个函数的参数。例如，调用 max 函数，求出三个数中的最大值，可以使用下列赋值语句完成。

```
x=max(max(a,b),c);
```

其中 max(a,b)作为函数 max(max(a,b),c)的一个参数使用。

8.3 数组作为函数参数

1. 数组元素作函数参数

数组元素作函数的实参，与用变量作实参一样，这时参数传递方式属于值传递方式。

例 8.7 求 10 个数中的最大值。

设：b——存放最大值。

程序：

```
/*求 10 个数中的最大值*/
#include <stdio.h>
main()
{int a[10],b,i;
 int max(int x,int y);
 printf("请输入 10 个数据到数组 a:\n");
 for (i=0;i<10;i++)
    scanf("%d",&a[i]);
 b=a[0];
 for (i=1;i<10;i++)
    b=max(b,a[i]);
 printf("最大值为: %d\n",b);
}
int max(int x,int y)                /*定义有参函数 max*/
{int z;                             /*函数中的说明部分*/
 if (x>y)
   z=x;
 else
```

```
  z=y;
 return(z);
}
```

2. 数组名作函数参数

数组名作函数参数，这种参数的传递方式属于地址传递方式。

如果形式参数是数组形式，则实际参数必须是实际的数组名，如果实际参数是数组名，则形式参数可以是同样维数的数组名或指针。必须在主调函数和被调函数中分别定义数组。实参数组和形参数组必须类型一致，形参数组可以不必指明长度。在 C 语言中，数组名除了作变量的标识符外，数组名还代表了该数组在内存中的起始地址，因此，当数组名作函数参数时，实参与形参之间是地址传递，实参数组名将该数组的起始地址传递给形参数组，两个数组共享相同的内存单元。编译系统不再为形参数组另外分配存储单元。

例 8.8　在一个一维数组中存放 10 个学生的 C 语言程序设计的成绩，求平均成绩。

程序：

```
/*数组名作为函数的参数——传地址*/
#include "stdio.h"
float average(float x[10]);
main()
{float a[10],aver;
 int i;
 printf("请输入 10 个学生的成绩:\n");
 for (i=0;i<10;i++)
    scanf("%f",&a[i]);
 aver=average(a);
 printf("平均分是:%5.1f",aver);
}
float average(float x[10])
{int i;
 float p,sum=0;
 for (i=0;i<10;i++)
    sum=sum+x[i];
 p=sum/10;
 return(p);
}
```

例 8.9　学校体育部对运动会已产生男子 100 米预赛和决赛成绩进行统计分析，需要将成绩排序。设计程序，将学校运动会的男子 100 米预赛和决赛成绩分别输入到数组 a 和数组 b，通过调用排序函数 sort，分别将预赛成绩和决赛成绩进行排序，最后输出排序后的结果。设有 32 名运动员参加预赛，8 名运动员参加决赛。

程序：

```
/*数组名作为函数的参数——传地址*/
#include "stdio.h"
main()
{float a[32],b[8];
 int i;
 void sort(float array[ ],int n);
 printf("请输入预赛成绩: \n");
 for (i=0;i<32;i++)
    scanf("%f",&a[i]);
 printf("请输入决赛成绩: \n");
 for (i=0;i<8;i++)
```

```
    scanf("%f",&b[i]);
 printf("\n");
 sort(a,32);
 printf("排序后的预赛成绩:\n");
 for (i=0;i<32;i++)
    printf("%-10.2f",a[i]);
 printf("\n");
 sort(b,8);
 printf("排序后的决赛成绩:\n");
 for (i=0;i<8;i++)
    printf("%-10.2f",b[i]);
 printf("\n");
}
void sort(float array[ ],int n)
{int i,j,k;
 float t;
 for (i=0;i<n-1;i++)
    {k=i;
    for (j=i+1;j<n;j++)
       if (array[j]<array[k]) k=j;
    if (i!=k)
    {t=array[k];array[k]=array[i];array[i]=t;}
    }
}
```

例 8.10 设计一个求两个 $N\times M$ 矩阵之和的函数 add，在主函数中调用 add 函数完成求矩阵 A 与矩阵 B 之和的功能。

程序：

```
#include "stdio.h"
#define N 3
#define M 4
main()
{
 int a[N][M],b[N][M],c[N][M];
 int i,j;
 void add(int a[N][M],int b[N][M],int c[N][M]);
 printf("Please input array a:\n");
 for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      scanf("%d",&a[i][j]);
 printf("Please input array b:\n");
 for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      scanf("%d",&b[i][j]);
 add(a,b,c);
 printf("Sum:\n");
 for (i=0;i<N;i++)
    {
     for (j=0;j<M;j++)
       printf("%-8d",c[i][j]);
     printf("\n");
    }
}
```

```
void add(int x[N][M],int y[N][M],int z[N][M])
{
 int i,j;
 for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      z[i][j]=x[i][j]+y[i][j];
}
```

程序中，主调函数使用了函数调用语句 add(a,b,c);来调用 add 函数，使用了数组名 a、b 和 c 来作三个实参，因此传递的是数组的地址。形参数组 x、y 和 z 接收到实参传递过来的地址后，分别与数组 a、b 和 c 共享内存空间。换句话说，在 add 函数中对数组 x、y 和 z 进行操作，实际上就是在对数组 a、b 和 c 进行操作。

多维数组名作实参时，在被调函数中对形参数组定义时应定义每一维的大小，可以省略第一维的大小说明，但是，不能省略其他维的大小。

8.4　函数的嵌套调用

一个函数调用另一个函数，这个函数又调用另外一个函数，这样的调用称为函数的嵌套调用。函数嵌套调用的示意图如图 8.6 所示。

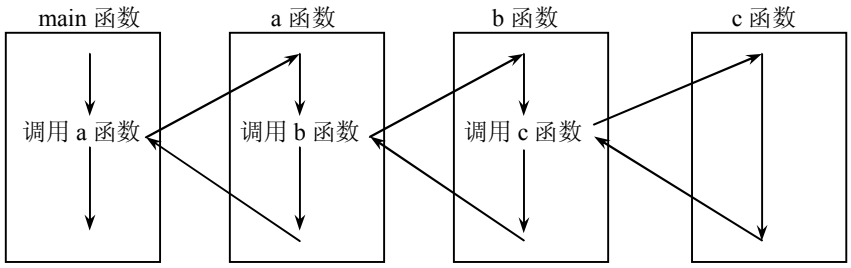


图 8.6　嵌套调用示意图

例 8.11　设计程序对学校运动会的男子 100 米预赛和决赛的成绩进行管理，管理系统中设计两个模块，一个模块对预赛成绩进行管理，另一个模块对决赛成绩进行管理。每个模块都有输入、排序和输出功能。要求将运动员编号和预赛成绩输入到二维数组 a，将运动员编号和决赛成绩输入到二维数组 b。通过调用排序函数 sort，分别将预赛成绩和决赛成绩进行排序，最后输出参加决赛的运动员名单和决赛成绩公告。设有 32 名运动员参加预赛，8 名运动员参加决赛，决赛成绩取前 7 名。

程序：

```
#include "stdio.h"
#define N 10
main()
{ float a[N][2],b[8][2];
  int i,m=1;
  void z1(float x[][2]);
  void z2(float x[][2]);
  void sort(float array[ ][2],int n);
  while (m!=3)
  {
   clrscr();                          /*清屏*/
```

```
   printf("********************\n");
   printf("1.男子100M预赛管理\n");        /*显示菜单*/
   printf("2.男子100M决赛管理\n");
   printf("3.退出\n");
   printf("********************\n");
   do                                    /*容错*/
   {
    printf("请按需输入(1~3):\n");
    scanf("%d",&m);
   } while (m>3 || m<1);
   if (m==1)
    z1(a);
   else if (m==2)
    z2(b);
  }
}
void z1(float x[][2])
{
  int i,j;
  float y;
  printf("请输入预赛成绩：\n");
  for (i=0;i<N;i++)
    for (j=0;j<2;j++)
       {scanf("%f",&y);
        x[i][j]=y;
       }
  sort(x,N);
  printf("男子100M参加决赛名单\n");
  printf("********************\n");
  printf(" 运动员号码  成绩\n");
  for (i=0;i<8;i++)
    printf("%8.0f%10.2f\n",x[i][0],x[i][1]);
  printf("按任意一键继续...\n");
  getch();
 }
 void z2(float x[][2])
 {
  int i,j;
  float y;
  printf("请输入决赛成绩：\n");
  for (i=0;i<8;i++)
    for (j=0;j<2;j++)
       {scanf("%f",&y);
        x[i][j]=y;
       }
  sort(x,8);
  printf("  男子100M成绩公告\n");
  printf("********************\n");
  printf(" 运动员号码  成绩\n");
  for (i=0;i<7;i++)
```

```
    printf("%8.0f%10.2f\n",x[i][0],x[i][1]);
  printf("按任意一键继续...\n");
  getch();
 }
 void sort(float array[ ][2],int n)
 {int i,j,k;
  float t;
  for (i=0;i<n-1;i++)
    {k=i;
     for (j=i+1;j<n;j++)
        if (array[j][1]<array[k][1]) k=j;
     if (i!=k)
     {t=array[k][0];array[k][0]=array[i][0];array[i][0]=t;}
     {t=array[k][1];array[k][1]=array[i][1];array[i][1]=t;}
     }
}
```

整个程序由 main 函数和三个自定义函数 z1、z2 和 sort 组成，z1 函数的功能是预赛成绩管理，z2 函数的功能是决赛成绩管理，sort 函数的功能是将成绩按从小到大的顺序排序。在程序中，main 函数调用 z1 函数，z1 函数又调用了 sort 函数，这就形成了函数的嵌套调用。同样地，main 函数调用 z2 函数，z2 函数又调用了 sort 函数，也是一个函数的嵌套调用。

如果将程序中多处出现的打印一行星号的功能设计为一个 printstar 函数供 z1 和 z2 函数调用，那么，也构成了函数的嵌套调用。

为了方便调试和运行程序，程序中使用了符号常量 N，并将常量定为 10。只需将常量 10 改写为 32，即可满足题目的要求。

8.5　函数的递归调用

8.5.1　递归的概念

通俗地讲，用自身的结构来描述自身，就称为递归。递归调用是指在调用一个函数的过程中，又出现直接地或间接地调用该函数本身，这种函数的调用称为递归调用。

例如：求 $n!$，可以表示为：

$$n!=n*(n-1)!$$
$$(n-1)!=(n-1)*(n-2)!$$

8.5.2　递归程序设计应用举例

例 8.12　用递归调用的方法求 $n!$。

可以将求 $n!$用以下式子来表示：

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

设计一个有参函数 fatco，其功能为求 n 的阶乘，形参 n 的值由主调函数提供，则可以使用以下式子表示求解过程：

$$\text{fatco}(n)=\begin{cases}1 & n=0\\ n*\text{fatco}(n-1) & n>0\end{cases}$$

程序：

```
#include <stdio.h>
main()
{int n;
 unsigned long int p;
 unsigned long int facto(int n);
 printf("n=");
 scanf("%d",&n);
 p=facto(n);
 printf("%d!=%ld\n",n,p);
}
unsigned long int facto(int n)
{unsigned long int r;
 if(n==0) r=1;
 else r=n*facto(n-1);
 return(r);
}
```

运行：

```
n=4↓
4!=24
```

运行：

```
n=0↓
0!=1
```

运行：

```
n=1↓
1!=1
```

运行：

```
n=5↓
5!=120
```

当 n=4 时，计算机的求解过程如下：

（1）facto(4)=4×facto(3)　　说明：n=4，调用 facto(3)

（2）facto(3)=3×facto(2)　　说明：n=3，调用 facto(2)

（3）facto(2)=2×facto(1)　　说明：n=2，调用 facto(1)

（4）facto(1)=1×facto(0)　　说明：n=1，调用 facto(0)

（5）facto(0)=1　　说明：n=0，求得 facto(0)的值

（6）facto(1)=1×1=1　　说明：回归，求得 facto(1)的值

（7）facto(2)=2×1=2　　说明：回归，求得 facto(2)的值

（8）facto(3)=3×2=6　　说明：回归，求得 facto(3)的值

（9）facto(4)=4×6=24　　说明：回归，求得 facto(4)的值

例 8.13　用递归调用的方法计算 x^n。

可以用以下式子来表示计算 x^n：

$$x^n=\begin{cases}1 & n=0\\ x\cdot x^{n-1} & n>0\end{cases}$$

设计一个有参函数 power，其功能为计算 x^n，该函数有两个形参，形参 x 和 n 的值由主调

函数提供，则可以使用以下式子表示求解过程：

$$\text{power}(x,n)=\begin{cases}1 & n=0\\ x*\text{power}(x,n-1) & n>0\end{cases}$$

程序：

```
/*递归求 x 的 n 次方*/
#include <stdio.h>
main()
{int x,n;
 long int power(int x,int n);
 printf("x=");
 scanf("%d",&x);
 printf("n=");
 scanf("%d",&n);
 printf("%d^%d=%ld\n",x,n,power(x,n));
}
long int power(int x,int n)
{
 if(n==0) return(1);
 else return(x*power(x,n-1));
}
```

运行：

```
x=3↓
n=4↓
3^4=81
```

运行：

```
x=2↓
n=3↓
3^4=8
```

运行：

```
x=10↓
n=8↓
10^8=100000000
```

运行：

```
x=-5↓
n=3↓
-5^3=-125
```

例 8.14　使用辗转相除法求两个正整数的最大公约数。

设计一个 gcd 函数，可以将辗转相除法表示为：

$$\text{gcd}(m,n)=\begin{cases}n & m\%n=0\\ \text{gcd}(n,m\%n) & m\%n\neq 0\end{cases}$$

程序：

```
/*递归求两数的最大公约数*/
#include "stdio.h"
main()
{int x,y,z;
 int gcd(int m,int n);
 printf("Please input x,y:");
 scanf("%d,%d",&x,&y);
 z=gcd(x,y);
```

```
 printf("%d 和%d 的最大公约数是%d\n",x,y,z);
}
int gcd(int m,int n)
{int a;
 if (m%n==0) a=n;
 else a=gcd(n,m%n);
 return(a);
}
```

运行：

```
Please input x,y:32,12↓
32 和 12 的最大公约数是 4
```

8.6 局部变量和全局变量

8.6.1 局部变量

在一个函数内部定义的变量称为局部变量。它只在本函数范围内有效，只有在本函数中可以使用，在此函数以外不能使用这些变量。

说明：

（1）主函数 main 中定义的变量只能在主函数中有效。

（2）不同函数中可以使用相同名字的变量，它们代表不同的对象，互不干扰。

（3）形式参数也属于局部变量。

（4）在一个函数内部，可以在复合语句中定义变量，这些变量只在本复合语句中有效。

在以前章节中所学习的变量以及函数的形参等都属于局部变量。

8.6.2 全局变量

在函数之外定义的变量称为外部变量，外部变量是全局变量。

全局变量可以为本文件中其他函数所共用。它的有效范围为从定义变量的位置开始到本源文件结束。

例 8.15 使用全局变量设计程序，通过调用函数实现交换两个变量值的功能。

程序：

```
#include "stdio.h"
int a,b;                        /*定义全局变量*/
main()
{void swip();                   /*函数原型*/
 printf("Input\n");
 printf("********\n");
 printf("a=");
 scanf("%d",&a);
 printf("b=");
 scanf("%d",&b);
 printf("********\n");
 swip();
 printf("Output\n");
 printf("********\n");
 printf("a=%d\nb=%d\n",a,b);
```

```
 printf("********\n");
}
void swip()              /*定义 swip 函数,功能是交换变量 a 和 b 的值*/
{int t;
 t=a;
 a=b;
 b=t;
}
```

运行：

```
Input
********
a=10000↓
b=30000↓
********
Output
********
a=30000
b=10000
********
```

全局变量增加了函数间数据联系的渠道，然而，全局变量的使用使得函数的通用性降低，因为函数在执行时要依赖于其所在的外部变量。过多使用全局变量，会降低程序的清晰性。另外，全局变量在程序的整个执行过程中都占用存储单元。

如果在同一个源程序文件中，外部变量与局部变量同名，则在局部变量的作用范围内，外部变量被“屏蔽”，因而不受干扰。

8.7　变量的存储类别

8.7.1　动态存储方式与静态存储方式

从变量的作用域（即从空间）来分，可以把变量分为全局变量与局部变量。从变量值存在的时间（即生存期）来分，可以把变量分为静态存储方式和动态存储方式。静态存储方式是指在程序运行期间分配固定的存储空间的方式。动态存储方式是指在程序运行期间根据需要动态地分配存储空间的方式。

内存中提供用户使用的存储空间示意图如图 8.7 所示，这个存储空间分为 3 个部分：

（1）程序区。

（2）静态存储区。

（3）动态存储区。

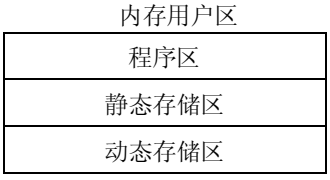


图 8.7　内存中提供用户使用的存储空间示意图

数据分别存放在静态存储区和动态存储区中。全局变量存放在静态存储区中，在程序开始执行时给全局变量分配存储区，程序执行完毕就释放这些空间。在程序执行过程中它们占用

固定的存储单元，而不是动态地进行分配和释放。

在动态存储区中存放以下数据：

（1）函数的形式参数。

（2）自动变量（未加 static 声明的局部变量）。

（3）函数调用时保护现场和返回地址等。

在 C 语言中数据的存储类别分为两大类：静态存储类和动态存储类。具体包含 4 种：自动的（auto）、静态的（static）、寄存器的（register）和外部的（extern）。根据变量的存储类别，可以确定变量的作用域和生存期。

8.7.2 auto 变量

用 auto 说明类别的变量或者在函数中未说明存储类别的局部变量称为自动变量。函数中的形式参数和在函数中定义的，包括在复合语句中定义的不加以类别说明的变量，都属于自动变量。自动变量是自动地分配存储空间的，数据存储在动态存储区中，在调用该函数时系统予以分配空间，在函数结束时就自动释放所占用的空间。

8.7.3 用 static 声明的局部变量

使用 static 说明其类别的局部变量称为静态局部变量。

静态局部变量在函数调用结束后，变量的值不消失，即不释放所占用的存储单元，在下一次调用函数时，静态局部变量保存着原来的值。在 C 语言中静态局部变量只被初始化一次。

可以对静态局部变量与自动局部变量作一个比较：静态局部变量属于静态存储类别，在静态区内分配存储单元，在程序整个运行期间都不释放所占用的空间；而自动变量属于动态存储类别，在动态区内分配存储单元，在函数调用结束后释放所占用的空间。静态局部变量只在编译时初始化一次，以后每次调用函数时不再初始化变量，而且函数调用结束后不释放所占用的存储空间，从而保留了上一次函数调用结束时的值；而自动变量不是在编译时赋初值，而是在调用函数时初始化变量，每调用一次函数就初始化一次，而且函数调用结束后，就释放了所占用的存储空间。当静态局部变量未初始化时，编译系统自动对数值型变量赋初值 0，对字符变量赋空值；而自动变量不初始化，则它的值是原来单元中残留的一个随机值。这是由于每次函数调用结束后存储单元已经释放，下次调用时又重新另外分配存储单元所造成的。

当需要保留函数上一次调用的值时，或是当初始化后，变量只被引用而不改变其值的时候使用静态局部变量较好。

例 8.16 分析下列 2 个程序，写出程序的运行结果。

程序 1：

```
/*静态变量 x*/
#include "stdio.h"
main()
{int i;
 void f();
 for (i=1;i<=6;i++)
 f();
 printf("\n");
}
void f()
{static int x=0;
```

```
 x=x+1;
 printf("x=%-4d",x);
}
```

运行：

```
x=1   x=2   x=3   x=4   x=5   x=6
```

程序 2：

```
/*自动变量 x*/
#include "stdio.h"
main()
{int i;
 void f();
 for (i=1;i<=6;i++)
 f();
 printf("\n");
}
void f()
{int x=0;
 x=x+1;
 printf("x=%-4d",x);
}
```

运行：

```
x=1   x=1   x=1   x=1   x=1   x=1
```

例 8.17　打印 1～5 的阶乘值。

程序：

```
int fac(int n)
{static int f=1;
 f=f*n;
 return(f);
}

#include "stdio.h"
main()
{int i;
 for (i=1;i<=5;i++)
 printf("%d!=%d\n",i,fac(i));
}
```

运行：

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

8.7.4　register 变量

为了提高程序执行效率，C 语言允许将局部变量的值存放在 CPU 的寄存器中，需要使用时直接从寄存器中取出数据参加运算，不必到内存中去存取。由于寄存器的存取速度远远高于内存的存取速度，因此可以提高程序的执行效率。

例 8.18　使用寄存器变量，分别求 1!、2!、3!、4!和 5!的值。

程序：

```
/*使用寄存器变量，打印 1～5 的阶乘值*/
int fac(int n)
{register int i,f=1;            /*定义寄存器变量*/
 for (i=1;i<=n;i++)
    f=f*i;
 return(f);
}
#include "stdio.h"
main()
{int i;
 for (i=1;i<=5;i++)
 printf("%d!=%d\n",i,fac(i));
}
```

运行：

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

只有局部自动变量和形式参数可以作为寄存器变量。在调用一个函数时占用一些寄存器以存放寄存器变量的值，在函数调用结束时释放寄存器。一个计算机系统中的寄存器数目极为有限，不能过多定义寄存器变量。当今的优化编译系统能够识别使用频繁的变量，从而自动地将这些变量放在寄存器中，而不需要程序设计者指定。局部静态变量不能定义为寄存器变量。

8.7.5 用 extern 声明的外部变量

外部变量即全局变量是在函数外部定义的，它的作用域为从变量的定义处开始，到本程序文件的末尾。在此作用域内，全局变量可以为程序中各个函数所引用。编译时将外部变量分配在静态存储区。有时需要用 extern 来声明外部变量，以扩展外部变量的作用域。

1. 在一个文件内声明外部变量

如果外部变量不在文件的开头定义，其有效的作用范围只限于定义处到文件结束。如果在定义点之前的函数想引用该外部变量，则应在引用之前用 extern 对该变量做外部变量声明。

例 8.19 用 extern 声明外部变量，扩展它在程序文件中的作用域。

程序：

```
/*用 extern 声明外部变量，扩展它在程序文件中的作用域。*/
#include "stdio.h"
main()
{int a,b,c;
 int max(int x,int y);          /*声明有参函数 max*/
 extern int A,B;                /*外部变量声明*/
 c=max(A,B);
 printf("max=%d\n",c);
}
int A=12,B=-8;                  /*定义外部变量*/
int max(int x,int y)            /*定义有参函数 max*/
{int z;                         /*函数中的说明部分*/
 if (x>y)
   z=x;
```

```
 else
   z=y;
 return(z);
}
```

运行:

```
max=12
```

2. 在多个文件程序中声明外部变量

如果一个程序包含两个以上的文件，在两个文件中都要用到同一个外部变量 *x*，那么，不能分别在两个文件中各自定义一个外部变量 *x*，否则在进行程序的链接时会出现“重复定义”的错误。在出现这种情况时，应在其中一个文件中定义外部变量 *x*，在另一个文件中用 extern 对 *x* 变量做外部变量声明。声明变量 *x* 的语句如下所示：

```
extern x;
```

例 8.20　将变量 *a* 和 *b* 定义为全局变量，设计程序，交换变量 *a* 和 *b* 的值。

文件 file1.c 内容如下所示：

```
/*文件名：file1.c*/
/*用 extern 将外部变量的作用域扩展到其他文件。*/
#include "stdio.h"
int a,b;                        /*定义外部变量*/
main()
{void swip();                   /*函数原型*/
 printf("a=");
 scanf("%d",&a);
 printf("b=");
 scanf("%d",&b);
 swip();
 printf("a=%d\nb=%d\n",a,b);
}
```

文件 file2.c 内容如下所示：

```
/*文件名：file2.c*/
/*用 extern 将外部变量的作用域扩展到其他文件。*/
extern a,b;                     /*声明 a 和 b 为已定义的全局变量*/
void swip()                     /*定义 swip 函数,功能是交换变量 a 和 b 的值*/
{int t;
 t=a;
 a=b;
 b=t;
}
```

当编译链接以上程序后便可以运行程序，运行结果如下所示：

```
a=8↓
b=5↓
a=5
b=8
```

在以上程序中，使用了全局变量 *a* 和 *b*，因此这两个变量的作用范围是整个程序，但由于这个程序是由 2 个源程序文件组成的，因此，在使用全局变量 *a* 和 *b* 时，应在一个源程序文件中定义，在另一个源程序文件中声明。以上的程序是在 file1.c 中定义变量 *a* 和 *b* 的，而在 file2.c 中对它们进行声明的。在下一节中将介绍如何将多个源程序文件构成的一个程序编辑链接成一个可执行文件的方法。

8.8 工程文件

有的程序是由多个源程序文件组成的，那么，如何组装这些文件呢？首先应创建一个工程文件，然后分别编译各个源程序，最后链接生成可执行程序。下面以例 8.19 为例介绍在 Turbo C 2.0（简称 TC）环境下组装 file1.c 和 file2.c 文件的具体步骤。操作步骤如下所示：

（1）创建工程文件。

File→Load→输入工程文件名

例如，要在 d 盘的 c 文件夹中创建名为 abc 的工程文件，则在弹出的输入文件名对话框中输入：d:\c\abc.prj。

（2）编辑工程文件。在 TC 的编辑窗口中输入各个文件名，如果文件不在系统默认的路径下，应当在文件前加上路径，并用 F2 保存，建成工程文件 abc.prj。

例如，输入：

file1.c

file2.c

（3）在 TC 主菜单中“Project”下，选择“Project name”菜单，输入工程文件名。

Project→Project name→输入 abc.prj

（4）编译源程序。分别编译 file1.c 和 file2.c，生成 file1.obj 和 file2.obj 文件。

Compile→compile to OBJ

（5）链接生成可执行程序 abc.exe。

Compile→Make EXE file

当完成了以上操作后，系统将生成与工程文件同名的可执行文件，例如 abc.exe。这时，可以运行可执行文件 abc.exe，得到程序的运行结果。

8.9 编译预处理

编译预处理是指在进行编译的第 1 遍扫描——词法扫描和语法分析之前所做的工作。这些预处理命令是由 ANSI C 统一规定的，但它不是 C 语言本身的组成部分。预处理是 C 语言的一个重要功能，它由预处理程序完成。当对一个源文件进行编译时，系统将自动引用预处理程序对源程序中的预处理部分进行处理，处理完成后自动开始编译，其示意图如图 8.8 所示。

图 8.8 预处理示意图

C 语言提供了多种预处理功能，如宏定义、文件包含及条件编译等。合理地使用预处理功能来编写程序，可以使程序便于阅读、修改、移植和调试，也有利于进行模块程序设计。本节将介绍常用的几种编译预处理命令。

8.9.1　宏定义

在 C 语言源程序中允许用一个标识符表示一个字符串，称为“宏”。被定义为“宏”的标识符称为“宏名”。编译预处理时程序中出现的所有宏名都用宏定义中的字符串来代换，称为“宏代换”或“宏展开”。宏代换是由预处理程序自动完成的。在 C 语言中，宏分为有参数和无参数两种，下面分别讨论这两种宏的定义和调用。

1. 无参数宏定义

无参宏的宏名后不带参数。其定义的一般形式为：

#define 宏名 替换文本

功能：使用**替换文本**所表示的字符串来替换从定义的位置开始到该源程序文件结束的程序中所出现的**宏名**。

其中，**宏名**用标识符表示，**替换文本**可以是任意字符串。前面介绍过的符号常量的定义就是一种无参宏定义。通常对程序中反复使用的表达式进行宏定义。例如使用了无参宏定义命令：

```
#define M 30
```

那么，预处理程序就将程序中所有的标识符 M 都用文字串 30 来代替。在预编译时将宏名替换成字符串的过程称为“宏展开”。

宏定义命令在编译预处理中只是一种简单的置换，但是对于程序中用双引号括起来的字符串不进行替换。**替换文本**所表示的字符串中可以含有任何字符，可以是常数，也可以是表达式，预处理程序对它不做任何正确性检查。如有错误，则只能在编译已被宏展开后的源程序时发现。

例 8.21　“宏展开”的应用举例。

```
#define M (x*x)
#include "stdio.h"
main()
{
 int s,x;
 printf("Please input:");
 scanf("%d",&x);
 s=7*M;
 printf("s=%d\n",s);
}
```

运行：

```
Please input:10↓
s=700
```

程序中首先进行宏定义，定义 M 为表达式 x*x，然后在 s=7*M;语句中进行宏调用。预处理时经宏展开后该语句变为：

```
s=7*(x*x);
```

因此，当输入整数 10 到变量 *x* 中时，程序的运行结果为 700。在宏定义时必须十分注意，应保证宏代换之后不会发生错误。

例 8.22　分析下列程序，写出程序的运行结果。

```
#define P 60
#include "stdio.h"
main()
{
```

```
 printf("P");
 printf("\n");
}
```

运行：

```
P
```

本例中定义宏名 P 表示 60，但在 printf 函数中 P 被双引号括起来了，因此不做宏代换。因此，程序的运行结果为 P，这表示把 P 当做字符串处理。

宏定义必须写在函数外，其作用域为宏定义命令起到源程序结束。如要终止其作用域，可使用#undef 命令，其语法为：

#undef 宏名

例如：

```
#define M 30                    /*定义 M*/
#undef M                    /*取消 M 的定义*/
```

宏定义允许嵌套，宏定义的字符串中可以使用已定义的宏名。宏展开时由预处理程序层层代换。

例 8.23 分析下列程序的运行结果。

```
#define PI 3.1415926
#define R 10
#define L 2*PI*R
#define S PI*R*R
#include "stdio.h"
main()
{
 printf("L=%f\nS=%f\n",L,S);
}
```

运行结果为：

```
L=62.831852
S=314.159260
```

经过宏展开后，printf 函数中的输出项 L 被展开为：

```
2*3.1415926*10
```

S 被展开为：

```
3.1415926*10*10
```

则语句

```
printf("L=%f\nS=%f\n", L,S);
```

变为

```
printf("L=%f\nS=%f\n",2*3.1415926*4.5,3.1415926*10*10);
```

2. 带参数宏定义

C 语言允许宏带有参数。宏定义中的参数称为形式参数，宏调用中的参数称为实际参数。对于带参数的宏，在调用过程中，不仅要宏展开，而且要用实参代换形参。

带参宏定义的一般形式为：

#define 宏名（形参表） 字符串

在**字符串**中应含有各个形参。带参宏调用的一般形式为：

宏名（实参表）

例如有下列宏定义：

```
#define PI 3.1415926            /*无参数宏定义*/
#define S(r) PI*r*r             /*带参数宏定义*/
```

使用语句

```
k=S(4.5);                    /*宏调用*/
```

调用这个宏时，用实参 4.5 代替形参 *r*，则经预处理宏展开后的语句为：

```
k=3.1415926*4.5*4.5;         /*宏展开*/
```

例 8.24　分析下列程序。

```
#include "stdio.h"
#define MAX(a,b) (a>b)?a:b
main()
{int x,y,max;
 printf("Please input x,y:\n");
 scanf("%d,%d",&x,&y);
 max=MAX(x,y);
 printf("max=%d\n",max);
}
```

运行：

```
Please input x,y:
8,3↓
max=8
```

程序的第 2 行是带参宏定义，用宏名 MAX 表示条件表达式(a>b)?a:b，形参 a 和 b 均出现在条件表达式中。语句 max=MAX(x,y);为宏调用，实参 x 和 y 将代换形参 a 和 b。宏展开后该语句为：

```
max=(x>y)?x:y;
```

该语句用于获得 *x*、*y* 中的较大者。

带参数宏定义与函数的含义是不同的，在带参数宏定义中，只是用实参替换形参，不存在参数的传递。

8.9.2　文件包含

文件包含是 C 语言预处理程序的另一个重要功能。文件包含的一般形式为：

#include "文件名"

或

#include <文件名>

功能：用**文件名**指定的文件内容来替换命令行，从而将指定文件包含到使用命令行的文件中来。如果文件名用尖括号括起来，则按系统指定的标准方式查找被包含文件；如果文件名用双引号括起来，则先在使用包含命令的源文件所在的目录中查找被包含的文件，如果没有找到，再按系统指定的标准方式检索其他目录。

前面已多次使用#include 命令来包含标准库函数的头文件。例如：

```
#include "stdio.h"
#include "math.h"
```

一个#include 命令只能包含一个文件，因此，如果要在一个源文件包含多个文件，则需要使用多个包含命令。一般在一个源程序文件的开始处会有多个包含命令。通常被包含的文件为头文件，也可以是 C 语言源程序文件，但是，包含 C 语言源程序文件时，在不同的编译系统中，其使用方法是不同的，例如在使用 Visual C++与使用 TC 时是不同的。显然，如果对被包含文件的内容进行了修改，则需要对包含该文件的源文件重新进行编译。另外，文件包含允许嵌套，即在一个被包含的文件中又可以包含另一个文件。

使用文件包含，可以将一个大的应用系统划分为一个个小模块，由不同的项目小组成员去同时进行研究与开发工作，便于实现模块化程序设计，提高工作效率。

8.9.3 条件编译

预处理程序提供了条件编译的功能。可以按不同的条件编译不同的程序部分，以及不同的目标代码文件。这对于程序的移植和调试是很有用处的。为了初学者方便学习，可以将条件编译概括为以下 3 种形式。

1. 形式 1——判断标识符是否已定义

#ifdef 标识符
　　程序段 1
#else
　　程序段 2
#endif

其功能是：如果**标识符**已被#define 命令定义过，则对**程序段 1**进行编译，否则对**程序段 2**进行编译。如果只有**程序段 1**而没有**程序段 2**，则形式 1 可简化为：

#ifdef 标识符
　　程序段
#endif

这里的**程序段**可以是语句组，也可以是命令行。这种条件编译对于 C 语言源程序的通用性是很有好处的。如果一个 C 语言源程序在不同的计算机系统上运行，而不同的计算机又有一定的差异（如有的计算机用 16 位存放一个整数，而有的则用 32 位存放一个整数），则往往需要对程序做必要的修改，这样就降低了程序的通用性。这时，可以使用以下的条件编译来处理。

```
#ifdef COMPUTER_A
  #define INTEGER_SIZE  16
#else
  #define INTEGER_SIZE  32
#endif
```

即如果 COMPUTER_A 在前面已被定义过，则编译命令行

```
#define INTEGER_SIZE  16
```

否则编译命令行

```
#define INTEGER_SIZE 32
```

这样，源程序就可以在不做任何修改的情况下用于不同的计算机系统。

2. 形式 2——判断标识符是否未定义

#ifndef 标识符
　　程序段 1
 # else
　　程序段 2
 # endif

与形式 1 的区别是将 ifdef 改为 ifndef。其功能是：如果标识符未被#define 命令定义过，则对**程序段 1**进行编译，否则对**程序段 2**进行编译。它与形式 1 的功能正好相反。

3. 形式 3——判断常量表达式是否为真（非 0）

#if 常量表达式

　　程序段 $_1$

#else

　　程序段 $_2$

#endif

其功能是：如果**常量表达式**为真（非 0），则对**程序段 $_1$** 进行编译，否则对**程序段 $_2$** 进行编译。因此可以在不同的条件下，使程序完成不同的功能。

当有多个分支的情况下，可使用以下一般形式：

#if 常量表达式 $_1$

　　程序段 $_1$

#elif 常量表达式 $_2$

　　程序段 $_2$

#elif 常量表达式 $_3$

　　程序段 $_3$

……

#elif 常量表达式 $_{n-1}$

　　程序段 $_{n-1}$

#else

　　程序段 $_n$

#endif

其中，**#elif** 意味着“else if”，形成一个“if-else-if”嵌套，从而构成多分支编译选择。其功能与“if-else-if”语句类似。预处理程序首先测试**#if** 中的**常量表达式 $_1$** 是否为真（非 0），只有为假（0）时，才对其后的**#elif** 中的**常量表达式 $_2$** 进行测试，……，依此类推，一旦测试到某个**常量表达式 $_i$** 为真，则编译**程序段 $_i$**，并且不再对其他**#elif** 后面的**常量表达式**进行测试。根据具体需求，可以使用也可以省略**#else** 和其后的**程序段 $_n$**。当使用**#else** 行时，则当所有**#elif** 之后的常量表达式都为假，即**常量表达式 $_1$**~**常量表达式 $_{n-1}$** 都为假时，则编译**程序段 $_n$**。

顺便指出，在形式 1 和形式 2 中介绍的**#ifdef** 和**#ifndef** 的条件编译中都可以使用**#elif**。

例 8.25　输入一个实数，根据需要设置条件编译，使之能以该实数为半径输出圆的面积，或以该实数为边长输出正方形的面积。

```
#define R 1
#include "stdio.h"
main()
{float c,r,s;
 printf("Please input:");
 scanf("%f",&c);
 #if R
    r=3.1415926*c*c;
    printf("area of round is:%f\n",r);
 #else
    s=c*c;
    printf("area of square is:%f\n",s);
 #endif
}
```

运行：

```
Please input:10↓
area of round is:314.159271
```

本例采用了第 3 种形式的条件编译。在程序第 1 行宏定义中，定义 R 为 1，因此在条件编译时，常量表达式的判定结果为真，则计算并输出圆面积。如果修改程序，将宏定义命令中的 R 定义为 0，这时，常量表达式的判定结果为假，则计算输出正方形的面积，运行结果为：

```
Please input:10↓
area of square is:100.000000
```

上面介绍的条件编译当然也可以用条件语句来实现。但是条件语句将会对整个源程序进行编译，生成的目标程序很长。而采用条件编译，则可根据条件只编译其中的**程序段 $_1$** 或**程序段 $_2$**，生成的目标程序较短。如果条件的程序段很长，则采用条件编译的方法是十分必要的。

下面对编译预处理作一个简单的小结：

（1）预处理功能是 C 语言特有的功能，它是在对源程序正式编译时由预处理程序完成的。程序员在程序中可用预处理命令调用这些功能。

（2）宏定义是用一个标识符来表示一个字符串，这个字符串可以是常量、变量或表达式。在宏调用中将用该字符串代换宏名。

（3）宏定义可以带有参数，宏调用时是以实参代换形参，而不是“值传送”。

（4）为了避免宏代换时发生错误，宏定义中的字符串应加圆括号，字符串中出现的形式参数两边也应加圆括号。

（5）文件包含是预处理的一个重要功能，可用于把多个源文件连接成一个源文件进行编译，结果将生成一个目标文件。

（6）条件编译允许只编译源程序中满足条件的程序段，使生成的目标程序较短，从而减少了内存的开销并提高了程序效率。

（7）使用预处理功能便于对程序进行修改、阅读、移植和调试，也便于实现模块化程序设计。

习题八

一、填空题

1．形参是指_____；实参是指_____。

2．局部变量是_____；全局变量是_____。

3．在 C 语言中，函数的形参隐含的存储类型说明是_____。

4．C 编译中的预处理是在编译_____进行的。

5．return 语句的一般形式是_____，其功能是_____。

6．C 语言中变量的存储类别有_____、_____、_____和_____。

二、单项选择题

1．C 语言规定，当简单变量作实参时，它和对应形参之间的数据传递方式为（　　）。

A．地址传递

B．单向值传递

C．由实参传给形参，再由形参传回实参

D．由用户指定传递方式

2．以下错误的描述是（　　）。

A．实参可以是常量、变量或表达式

B．形参可以是常量、变量或表达式

C．实参可以是任意数据类型

D．形参应与其对应的实参类型一致

3．C 语言中一个函数返回值的类型由（　　）决定。

A．return 语句中的表达式类型

B．调用函数的主调函数类型

C．系统默认的类型

D．定义函数时所指定的类型

4．有一个如下定义的函数，则该函数的类型为（　　）。

```
f(int a)
{printf("%d",a);
}
```

A．与参数 a 的类型相同　　B．void 的类型

C．没有返回值　　D．无法确定

5．在 C 语言程序中（　　）。

A．函数的定义可以嵌套，但函数的调用不可以嵌套

B．函数的定义不可以嵌套，但函数的调用可以嵌套

C．函数的定义和函数的调用不可以嵌套

D．函数的定义和函数的调用均可以嵌套

6．关于 return 语句，下列正确的说法是（　　）。

A．在主函数和其他函数中均可出现

B．必须在每个函数中出现

C．可以在同一个函数中出现多次

D．只能在除主函数之外的函数中出现一次

7．数组名作为实参传递给函数时，数组名被处理为（　　）。

A．该数组的长度

B．该数组的元素个数

C．该数组的首地址

D．该数组中各元素的值

8．下列哪种数据不存放在动态存储区中？（　　）。

A．函数形参变量

B．局部自动变量

C．函数调用时的现场保护和返回地址

D．局部静态变量

9．在一个源程序文件中，定义的全局变量的作用域为（　　）。

A．本文件的全局范围

B．本程序的全局范围

C．本函数的全局范围

D．从定义该变量的位置开始至本文件结束为止

10．以下程序的输出结果是（　　）。

```
long fun(int n)
{
 long s;
 if (n==1||n==2)
 s=2;
 else
 s=n-fun(n-1);
 return s;
}
main()
{
 printf("%ld\n",fun(3));
}
```

A．1　　B．2　　C．3　　D．4

三、阅读程序题

1．写出下列程序的运行结果。

```
#include "stdio.h"
main()
{int x=2,y=3,z=0;
 int func(int x,int y,int z);
 printf("(1) x=%d,y=%d,z=%d\n",x,y,z);
 func(x,y,z);
 printf("(3) x=%d,y=%d,z=%d\n",x,y,z);
}
int func(int x,int y,int z)
{z=x+y;
 x=x*x;
 y=y*y;
 printf("(2) x=%d,y=%d,z=%d\n",x,y,z);
}
```

2．写出下列程序的运行结果。

```
main()
{int i,a=3;
 for (i=0;i<3;i++)
 printf("%d,%d\n",i,f(a));
}
f(int a)
{auto int b=0;
 static int c=3;
 b++;c++;
 return(a+b+c);
}
```

3．写出下列程序的运行结果，并写出程序的功能。

```
#include "stdio.h"
int sum(int k);
main()
{int s,i;
 for (i=1;i<=10;i++)
```

```
   s=sum(i);
 printf("s=%d\n",s);
}
int sum(int k)
{static int x=0;
 return x+=k;
}
```

4．写出下列程序的运行结果。

```
#include "stdio.h"
void p1();
void p2();
static int a=5;
main ( )
{
 printf("a=%d\n",a);
 p1();
 p2();
}
void p1()
{
 printf("a*a=%d\n",a*a);
}
void p2()
{
 printf("a*a*a=%d\n",a*a*a);
}
```

四、程序填空题

1．以下函数的功能是求 x 的 y 次方。

```
double fun(double x,int y)
{
 int i;
 double z=1.0;
 for (i=1;i  (1)  ;i++)
    z=  (2)  ;
 return z;
}
```

2．以下函数的功能是使用函数递归调用的方法计算 $\sum_{i=1}^{n} i$。

```
sum(int n)
{int s;
 if (n<=0)
    printf("data error!\n");
 else if (n==1)
  (1)  ;
 else
  (2)  ;
 return s;
}
```

3．以下程序的功能是使用调用函数的方法计算 $s=1+2+2^2+2^3+\cdots+2^{10}$，程序中的自定义函数 func 的功能是计算 2 的 n 次方幂。

```
#include "stdio.h"
main()
{int func(int n);
 int s,i;
   (1)  ;
 for (i=0;i<=10;i++)
    s=  (2)  ;
 printf("%d\n",s);
}
int func(int n)
{int k,p;
 p=  (3)  ;
 for (k=1;k<=n;k++)
    p=  (4)  ;
 return  (5)  ;
}
```

五、编程题

1．计算组合$C_n^m=\frac{n!}{m!(n-m)!}$。设 m=6，n=10，m 和 n 的值要求使用键盘输入，求出结果后按下列格式输出，其中，在？号的位置上显示计算结果。要求使用一个自定义函数求出 p! 供主函数调用，另外使用一个自定义函数打印一行 “*”号供主函数调用。

****

C= ?

****

2．设计一个自定义函数 rep，功能为重复输出 n 个字符，然后在主函数里调用该函数，分别输出下列图形：

（1）**********
&&&&&&&&&&
****
&&&&&&&&&&
****
&&&&&&&&&&

（2）#
##
###
####
#####

（3）1
22
333
4444
55555

（4）　*
　　***
　　*****
　　*******
　　*********

3．设计一个计算字符串长度的自定义函数 len，调用该函数求一个字符串的长度。

4．计算 $s=1!+2!+3!+\cdots+10!$，要求使用一个自定义函数 facto 求出 $n!$供主函数调用，分别使用以下 3 种方法设计 facto 函数。

（1）使用自动变量与循环的方法。

（2）使用函数递归调用的方法。

（3）使用静态变量的方法。

5．函数嵌套调用练习。学生要在 6 门课程中选修 2 门课程，其中有 2 门是专业选修课程，要求在所选择的 2 门课程中，至少有一门是专业选修课。设计程序计算有多少种选法，要求设计一个函数 cnm 计算 C_n^m，设计一个函数 facto 计算 $p!$（提示：方法 1：$s=C_2^1C_4^1+C_2^2$；方法 2：$s=C_6^2-C_4^2$）。

6．函数递归调用练习。设计程序，根据组合的性质计算组合 $C_n^m=C_{n-1}^m+C_{n-1}^{m-1}$。

7．设要将一段英文书信译成密码，密码的编码规则是：用原来的字符后面第 4 个字母代替原来的字母。例如“A”后面的第 4 个字母是“E”，如此类推。设计一个程序，能够实现加密和解密功能。要求首先显示一个菜单供用户选择，当用户选择 1 时，实现加密功能，而当用户选择 2 时，实现解密功能。程序使用户通过键盘给字符数组输入要加密或者解密的文字，然后对文字进行处理，最后将加密或解密的文字显示出来。要求显示的功能菜单及提示信息如下：

功能菜单

```
**********************************
*              1. 加密             *
*              2. 解密             *
*              3. 退出             *
**********************************
Please input your choice:
```

8．为小学生设计一个验算程序，具有验算两个整数加、减、乘、除的功能。要求显示菜单如下，供小学生选择。当选择 1 时，让学生输入被加数和加数，然后将求和结果显示出来；当选择 2、3 或者 4 时，则做相应的操作。要求将输入与输出分别设计为输入函数 input 和输出函数 output。

功能菜单

```
**********************************
*            1. 加法验算           *
*            2. 减法验算           *
*            3. 乘法验算           *
*            4. 除法验算           *
**********************************
请输入（1~4）:
```