第 7 章 函数

在本章中，我们将介绍有关函数的知识。函数是结构化程序设计中的基本组成单位，因此掌握函数的定义和应用非常重要。

7.1 函数概述

在第 1 章中已经介绍过，C 程序是由函数组成的。一个较大的程序一般分为若干个以函数为单位的程序模块（或叫子程序），每个模块用来实现一个特定的功能。一个 C 程序可由一个主函数 main()和若干个其他函数组成，主函数可以调用其他函数，其他函数之间也可相互调用。由于采用了函数模块式的结构，C 语言易于实现结构化程序设计。可以使程序的层次结构清晰，便于程序的编写、阅读、调试。

虽然在前面各章的程序中都只有一个主函数 main()，但在本章中将讲述用多个函数编程。首先来看下面的例子。

【例 7.1】通过编程求两个数中的最大值，并且输出的最大值上下分别有一行星号。

```
#include <stdio.h>
void star()                    /* star 为输出一行星号函数*/
{printf("*************\n"); /*调用库函数 printf 输出一行星号*/
}
void main()
{int x,y,z;
 x=6;
 y=3;
 int max(int,int);             /*对 max 函数进行说明*/
 z=max(x,y);                   /*调用 max 函数，把实际参数 x,y 传给形式参数 a，b，将返
                                 回的最大值赋给 z*/
 star();                       /*调用 star 函数，输出一行星号*/
 printf("max is %d\n",z);      /*调用库函数 printf 输出最大值 z*/
 star();                       /*调用 star 函数，输出一行星号*/
}
int max(int a,int b)           /* max 为求最大值函数，a，b 为形式参数*/
{int c;
 if(a>b)
   c=a;
 else
   c=b;
 return (c);                   /*将最大值 c 返回 main 主函数*/
}
```

程序运行结果为：

```
"*************
max is 6
"*************
```

分析：在例 7.1 中用了 4 个函数，其中一个是主函数 main()，一个是库函数 printf，还有两个自己定义的函数：一个为求最大值函数 max(int a,int b)，另一个为输出一行星号函数

star()。由此程序可以引申出 C 程序的分类及基本概念。

在 C 语言中可从不同的角度对函数进行分类。

（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。

① 库函数

由 C 系统提供，用户无须定义，也不必在程序中作类型说明，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。例如 printf、scanf、getchar、putchar、gets、puts、strcat 等函数均属此类。C 语言提供了极为丰富的库函数，对于库函数读者可根据需要查阅有关手册。

② 自定义函数

由用户按需要编写的函数。对于自定义函数，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）从函数的返回值角度看，又可把函数分为有返回值函数和无返回值函数两种。

① 有返回值函数

此类函数被调用执行完后将向调用者返回一个执行结果，称为函数返回值。例如 int max(int a,int b)函数。

② 无返回值函数

此类函数用于完成某项特定的处理任务，执行完成后不向调用者返回函数值。由于函数无须返回值，用户在定义此类函数时可指定它的类型为“空类型”，空类型的说明符为“void”，例如 void star()。

（3）从主调函数和被调函数之间数据传送的角度看又可分为无参函数和有参函数两种。

① 无参函数

函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送，例如 star()。

② 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。例如 max 函数，定义时为 int max(int a,int b)，调用时为 max(x,y)。

7.2 函数的定义

用户自定义函数与变量一样，必须先定义，然后才能使用。在 C 语言中，所有的函数定义，包括主函数 main 在内，都是平行的。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。

7.2.1 函数定义的格式

函数定义的一般形式如下。

格式：

```
[类型说明符] 函数名([形式参数说明列表])
{
```

```
    类型说明部分
    语句部分                    （函数体部分）
    [return 表达式]
}
```

说明：

（1）类型说明符：说明函数名的类型。函数通过函数名返回一个值，因此函数名需要说明数据类型。类型说明符省略时函数类型为整型，也可用类型说明符“void”说明函数无返回值。

（2）函数名：函数名的命名规则与变量的命名规则相同。在一个程序中不同的函数其名称不能相同。

（3）形式参数说明列表：形式参数说明列表是用于调用函数和被调函数之间进行数据传递的，使用时需要在表中进行类型说明。其格式如下：

格式：

```
类型说明符 1  参数 1，类型说明符 2  参数 2，……，类型说明符 n  参数 n
```

形式参数说明列表省略时为无参函数，也可用“void”进行无参说明。例如，void star(void)为无参数传递且无返回值函数。形式参数简称为形参。

（4）return 表达式：将表达式的值赋值给函数名，返回到主调函数。可有多条 return 语句，也可省略 return 语句，省略时为无返回值。return 表达式可以是单独变量，常量，也可以是复杂的表达式；表达式外可以加圆括号，也可以不加。例如，return x，return (x)，return 56, return a+b。

（5）包含在{}内部的部分称为函数体，由说明部分和语句部分组成。

【例 7.2】编写求 n!的函数。

```
#include <stdio.h>
int fact(int n )          /* fact 为求 n!函数，n 为形式参数*/
{int i,s;
 for(i=1,s=1;i<=n;i++)
   s=s*i;                 /*计算 n!值*/
 return s;                /*将 n!值 s 赋给函数名 fact，带回主函数*/
}
```

（6）没有任何操作内容的函数称为空函数。在程序设计中将准备扩充功能的地方写一个空函数，该函数什么也不做，先占一个位置，在程序需要扩充功能时，再用一个已编好的有用函数取代它。例如：

```
函数名()
 { }
```

注意：对于无参函数或空函数，函数名后面的圆括号不能省略。

7.2.2　函数的说明（声明）

对于用户自定义函数，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说明，然后才能使用。在主调函数中对被调函数进行说明的目的是使编译系统知道被调函数返回值的类型，以便在主调函数中按此种类型对返回值做相应的处理。

被调函数的说明有两种格式，其一般形式为：

格式：

```
类型说明符 被调函数名(类型说明符 1 [形参 1], 类型说明符 2 [形参 2] …);
```

对于被调函数的说明，括号内既可以给出形参的类型和形参名，也可以省略形参名，只给出形参类型。例如，引例中对 max 函数的说明可写为：

```
int max(int,int);
```

也可写为：

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

关于是否对被调函数进行说明的几种情况：

（1）如果被调函数的定义是在主调函数之后，则必须在调用该被调函数之前对该被调函数进行说明，例如，例 7.1 中在主函数 main()内部，对 max 函数的说明。

（2）若被调函数的定义在主调函数之前，则可省略对该被调函数进行说明，例如，例 7.1 中的 star()。

（3）如在所有函数定义之前，在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数进行说明。例如：

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

其中第 1、2 行对 str 函数和 f 函数预先做了说明，因此在以后各函数中无须对 str 和 f 函数再做说明就可直接调用。

（4）对库函数的调用不需要再做说明，但必须把该函数的头文件用#include 命令包含在源文件前部，即是对相应函数库中的所有函数进行了说明。

7.3 函数的调用

当函数定义及函数类型说明过以后，就可以在主调函数中调用该函数，并通过对函数的调用来执行函数体，完成相应的功能。函数虽然不能嵌套定义，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。函数还可以自己调用自己，称为递归调用。main 函数是主函数，它可以调用其他函数，而不允许被其他函数调用。因此，C 程序的执行总是从 main 函数开始，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。一个 C 源程序必须有也只能有一个主函数 main。

7.3.1 函数调用的格式

函数调用的一般形式如下。

格式：

```
函数名([实际参数表])
```

说明：

（1）实际参数简称实参，省略实际参数表为无参函数调用。

（2）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。

7.3.2　函数调用的方式

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。

例如例 7.1 中的 z=max(x,y)语句，是一个赋值表达式，把 max 的返回值赋予变量 z。

2．函数语句

函数调用的一般形式加上分号即构成函数语句。

例如例 7.1 中的 star(); 都是以函数语句的方式调用函数。

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。

例如在例 7.1 中可以省略变量 z，直接用 printf("max is %d",max(x,y))语句输出函数返回值; 也就是把 max 调用的返回值又作为 printf 函数的实参来使用。

【例 7.3】在例 7.1 基础上，编程求任意 3 个数的最大值，并输出最大值。

```
#include <stdio.h>
void star()
{printf("*************\n");
}
int max(int a,int b)
{int c;
 if(a>b)
   return a;     *将最大值 a 返回 main 主函数*/
 else
   return b;     *将最大值 a 返回 main 主函数*/
}
void main()
{int x,y,z;
 printf("input number x:\n");
 scanf("%d",&x);
 printf("input number y:\n");
 scanf("%d",&y);
 printf("input number z:\n");
 scanf("%d",&z);
 star();
 printf("max is %d\n",max(max(x,y),z));  /*函数 max(x,y)为一个实际参数*/
 star();
}
```

程序运行过程与结果为：

```
input number x:
5↙
input number y:
9↙
input number z:
3↙
"*************
max is 9
"*************
```

分析：上题中利用了求两个数中的最大值函数，其调用方法为 printf("max is %d", max(max(x,y),z), 函数 max(x,y)的返回值作为函数 max(max(x,y),z)的一个实际参数，返回的是 x、y、z 这 3 个数中的最大值。

7.4 函数参数传递和函数的值

函数应用最重要的部分就是函数的参数传递和函数的返回值。

7.4.1 函数参数传递

前面已经介绍过，函数的参数分为形参和实参两种。形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。形参和实参的功能是传送数据。当发生函数调用时，主调函数把实参的值传送给被调函数的形参，从而实现主调函数向被调函数的数据传送。这样的参数传递方式叫做“数值传递”，简称“值传递”。

首先看一个例子。

【例 7.4】在例 7.2 的基础上，编程求任意 3 个数的阶乘和。

```
#include <stdio.h>
int fact(int n )
{int i,s;
 for(i=1,s=1;i<=n;i++)
   s=s*i;
 return s;
}
void main()
{int x,y,z;
 long ss;
 printf("input number x:\n");
 scanf("%d",&x);
 printf("input number y:\n");
 scanf("%d",&y);
 printf("input number z:\n");
 scanf("%d",&z);
 ss=fact(x)+fact(y)+fact(z);  /*计算阶乘和*/
 printf("*************\n");
 printf("x!+y!+z! is %ld\n",ss);
```

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

程序运行过程与结果为：

```
input number x:
3↙
input number y:
5↙
input number z:
7↙
"*************
x!+y!+z! is 5166
"*************
```

分析：

（1）在此例中，语句 ss=fact(x)+fact(y)+fact(z)多次调用了同一个函数 fact，只不过传递的参数不同，计算出了 x，y，z 的阶乘和。

（2）在此例中，语句 ss=fact(x)+fact(y)+fact(z)调用函数的顺序为：fact(x)、fact(y)、fact(y)；调用函数 fact(x)时，参数传递的顺序为：首先将实参 x 传给形参 n，然后在函数 fact(n)中计算 n!，接下来将 n! 值 s 赋给函数名 fact(x)。函数 fact(y)和函数 fact(z)的参数传递过程同理。函数调用和参数传递顺序如图 7-1 所示。

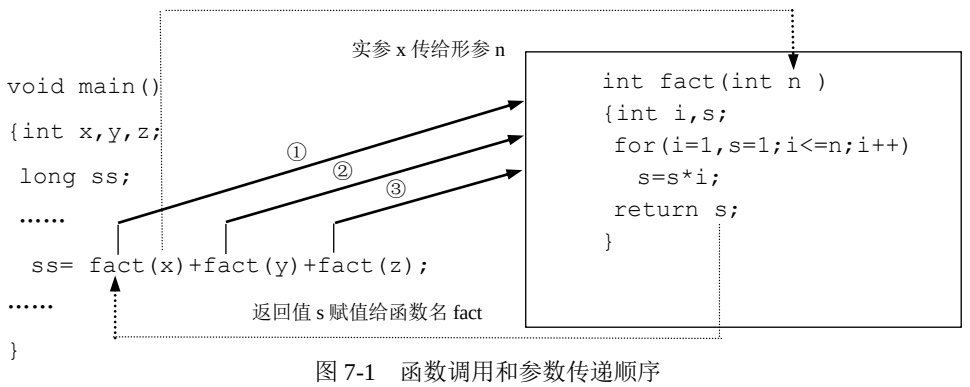


图 7-1　函数调用和参数传递顺序

关于函数形参和实参的说明：

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

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

（3）实参和形参在数量上、类型上、顺序上应严格一致，否则会产生“类型不匹配”的错误。

（4）函数调用中发生的数据传送是单向的，即只能把实参的值传送给（赋值给）形参，而不能把形参的值反向地传送给实参，因此在函数调用过程中，若形参的值发生改变，则不 会

影响实参中的值发生变化。例 7.5 可以说明这个问题。

【例 7.5】编程求 $n+(n-1)+(n-2)+\cdots+1$ 的和。

```
#include <stdio.h>
void main()
{int n;
 void s(int);                  /*调用前，说明函数 s*/
 printf("input number n\n");
 scanf("%d",&n);
 printf("*********\n");
 printf("n=%d\n",n);           /*调用函数 s 之前，输出 n 值*/
 printf("*********\n");
 s(n);
 printf("*********\n");
 printf("n=%d\n",n);           /*调用函数 s 之后，输出 n 值*/
 printf("*********\n");
}
void s(int n)
{int i;
 for(i=n-1;i>=1;i--)
   n=n+i;                      /*计算 n+(n-1)+(n-2)+…+1 的和*/
 printf("n=%d\n",n);           /*在函数 s 内部，输出 n 值*/
}
```

程序运行过程与结果为：

```
input number n:
5↙
*********
n=5                            /*调用函数 s 之前，输出 n 值*/
*********
n=15                           /*在函数 s 内部，输出 n 值*/
*********
n=5                            /*调用函数 s 之后，输出 n 值*/
*********
```

分析：

（1）本程序中定义了一个函数 s，该函数的功能是求 1+2+3+…+n 的和。在主函数中输入 n 值，并作为实参，在调用时传送给 s 函数的形参 n（注意，本例的形参变量和实参变量的标识符都为 n，但这是两个不同的变量，各自的作用域不同）。

（2）在主函数中分别在调用函数 s 之前和之后，用 printf 语句输出两次 n 值，这个 n 值是实参 n 的值。在函数 s 中也用 printf 语句输出了一次 n 值，这个 n 值是形参最后取得的 n 值。

（3）从运行情况看，输入 n 值为 5。即实参 n 的值为 5，第一次输出的是实参 n=5。把此值传给函数 s 时，形参 n 的初值也为 5，在执行函数过程中，形参 n 的值变为 15，第二次输出的是形参 n=15。返回主函数之后，第三次输出的仍然是实参 n=5。可见，虽然形参的值由 5 变为 15，但实参的值不随形参的变化而变化，始终是 5。

（4）在函数调用时，若存在多个参数的情况，实参表中各参数传递顺序是自左至右使用还是自右至左使用，各系统的规定不一定相同。在 Visual C++环境中，参数传递顺序是自右至左使用。见例 7.6。

【例 7.6】实参表中参数传递顺序问题。

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

例 7.6 的运行结果为：

```
8   7   8   8
```

7.4.2　函数的值

函数的值是指函数被调用之后，执行函数体中的程序段所取得的并要返回给主调函数的值。该值通过 return 语句将返回值赋值给函数名，由函数名带回主调函数。

如例 7.4 中调用 fact(x)函数计算 x 的阶乘值，通过函数名将阶乘值带回主调函数，用语句 ss=fact(x)+fact(y)+fact(z)通过 3 次调用 fact()函数实现计算阶乘和。

关于函数值（或称函数返回值）的说明：

（1）函数的值只能通过 return 语句返回主调函数。return 语句的一般形式为：

```
return 表达式;
```

或者为：

```
return (表达式);
```

该语句的功能是计算表达式的值，并返回给主调函数。在函数中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，因此只能返回一个函数值。

（2）函数值（通过 return 语句返回的值）的类型和函数定义中函数的类型应保持一致。如果两者不一致，则以函数类型为准，自动进行类型转换。

（3）如函数值为整型，在函数定义时可以省去类型说明。

（4）不返回函数值的函数，可以明确定义为“空类型”，类型说明符为“void”。如例 7.5 中函数 s 并不向主函数返回函数值，因此可定义为：

```
void s(int n)
{ …… }
```

一旦函数被定义为空类型后，就不能在主调函数中使用被调函数的函数值了。例如，在定义 s 为空类型后，在主函数中写下述语句 sum=s(n); 就是错误的。为了使程序有良好的可读性并减少出错，凡不要求返回值的函数都应定义为空类型。

【例 7.7】有关函数返回值的例子，比较两个字符的大小，并返回不同的值。

```
#include "stdio.h"
comp(char c1,char c2)
{ float k;
  if(c1>c2)
     return k=1.0;
  else if(c1<c2)
     return k=-1.0;
```

```
  else
    return k=0.0;
}
void main()
{  char s1,s2;
   int z;
   scanf("%c\n",&s1);
   scanf("%c",&s2);
   z=comp(s1,s2);
   if(z>0)
       printf("s1>s2\n");
   else if(z<0)
       printf("s1<s2\n");
   else
       printf("s1=s2\n");
}
```

程序运行过程与结果为：

```
x↙
p↙
s1>s2
```

分析：

（1）本例中函数 comp 未定义类型，则函数值默认为整型（第 3 点）。return 语句的返回值 k 为实型，二者的数据类型不同，按上述第 2 点，先将 k 转化为整型，然后将值赋给函数名 comp，返回到主调函数 main。

（2）本例利用了多条 return 语句（第 1 点），但只能返回一个值。

（3）单个字符变量作为参数传递，也是数值传递。

7.5 数组参数的传递

数组像普通变量一样可以作为函数参数使用，并进行数据传送。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。

7.5.1 数组元素作为函数参数

数组元素就是带下标的变量，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 7.8 说明了这种情况。

【例 7.8】编程判别一个整数数组中各元素的正负，若为正或零，输出 1；若为负输出−1，并计算正数的个数，输出数组元素的值及正数的个数。

```
#include <stdio.h>
```

```
int k=0;
void nzp(int x)
{if(x>=0)
  { x=1;
   printf("x is positive   %d\n",x);
    k++;
  }
  else
  { x=-1;
   printf("x is minus     %d\n",x);
  }
}
main()
{
  int a[5],i;
  printf("input 5 numbers:\n");
  for(i=0;i<5;i++)
  { scanf("%d",&a[i]);
    nzp(a[i]);
  }
  for(i=0;i<5;i++) printf("%4d",a[i]);
  printf("\npositive numbers count:%d\n",k);
}
```

程序运行过程与结果为：

```
input 5 numbers:
3↙
x is positive   1
-7↙
x is minus     -1
5↙
x is positive   1
-9↙
x is minus     -1
6↙
x is positive   1
   3  -7   5  -9   6
positive numbers count:3
```

分析：

（1）此程序首先定义一个无返回值函数 nzp，并说明其形参 x 为整型变量。在函数体中对所传过来的数组元素值，即 x 的值进行判断。若为正或零，使 x=1，并输出 x；若为负，使 x=-1，并输出 x。

（2）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实参调用一次 nzp 函数，即把 a[i]的值传送给形参 x，供 nzp 函数使用。第二个 for 语句的作用是在完成所有调用 nzp 函数返回主函数后，输出数组的各元素值。由结果可以看出，虽然形参 x 的值发生了变化，但并没有影响实参数组元素 a[i]的变化。

（3）在所有函数上面定义了一个整型变量 k，该变量为全局变量，它的作用范围是在它

定义位置以下的所有函数中都有效。因此在函数 nzp 中，计算出正数个数 k，然后在主函数中输出 k 的值。

7.5.2 数组名作为函数参数

由上节可知，用数组元素作实参时，对数组元素的处理是按普通变量对待的，实参向形参传递的方式是单向的“数值传递”。用数组名作函数参数时，不是采用的“数值传递”方式，而是“地址传递”方式。

1. “数值传递”方式与“地址传递”方式的比较

下面来对比分析两种传递方式。

- “数值传递”方式：形参变量和实参变量都是普通变量，由编译系统分配两个不同的内存单元，在单元内存储的是赋给形参变量或实参变量的值。在函数调用时发生的值传递是把实参变量的值赋给形参变量。数值传递的特点是形参的变化不能改变实参的变化。
- “地址传递”方式：形参变量和实参变量都是地址型的变量，由编译系统分配两个不同的内存单元，在单元内存储的是某内存单元的地址。在函数调用时发生的地址传递是把实参变量中所存储的某内存单元的地址赋值给形参变量。地址传递的特点是指由实参变量中所存储的地址而标定的内存单元中的数值会随着形参变量中所存储的地址而标定的内存单元中数值的变化而变化。

地址变量又称做“指针”，关于指针的详细内容将在第 8 章中介绍。

【例 7.9】数值传递。

```
#include <stdio.h>
void p(int a)
{ a=a+5;
}
void main()
{ int x=10;
  p(x);
  printf("x=%d\n",x);
}
```

程序运行结果为：

```
x=10
```

【例 7.10】地址传递。

```
#include <stdio.h>
void p(int *px)          /* 定义 px 为地址变量*/
{                        /* *px 表示所标定的内存单元里的值*/
  *px=*px+5;             /* px 所标定内存单元里的值加 5*/
}
 void main()
{int x=10;
```

```
  p(&x);                    /*调用函数 p，将 x 的地址传给 px*/
  printf("x=%d\n",x);
}
```

程序运行结果为：

```
x=15
```

分析：由运行结果和图 7-2 可以看出，例 7.9 为单向的“值传递”，当把实参 x 的值“10”赋值给形参 a 后，由于 x 和 a 是两个变量，因此 a 的变化不会影响 x 的变化。例 7.10 为“地址传递”，当把实参 x 的地址“&x”赋值给形参 px 后，由于&x 和 px 都是 x 的地址，存储 x 的内存单元是&x 和 px 所标定的同一个内存单元，*px 代表的就是 x 的值，因此*px 的变化就是 x 的变化。

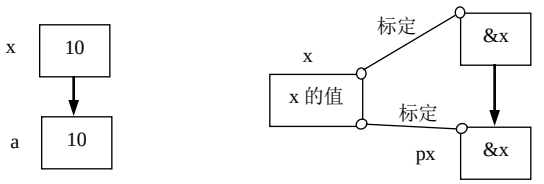


图 7-2　数值传递与地址传递的对比图

2．数组名作为函数参数

“地址传递”方式要求形参和相对应的实参都必须是类型相同的数组名或地址变量（指针变量）。本节只介绍数组名作函数参数的情况，其他详见第 8 章指针。

前面介绍过，用数组名作函数参数时，参数的传递是“地址传递”，那么，数据的传送是如何实现的呢？由数组一章我们知道，数组名代表了数组的首地址。在参数传递的时候，实参数组名把实参数组的首地址赋值给形参数组名，换句话说，形参数组名也代表了实参数组的首地址。在 C 语言中，实际上编译系统不为形参数组分配内存，形参数组与实参数组占用同一段内存空间，实参数组的数组值就是形参数组的数组值，就像一个人有两个名字一样。若形参数组的数组值发生变化，则实参数组的数组值也要发生相同变化，因为实际上他们就是一个数组的两个名字，可以见图 7-3。

【例 7.11】数组 a 中存放了 10 个数，用选择法对数组 a 中的元素进行升序排序，并输出。

```
#include <stdio.h>
void sort(int b[],int n)
{
   int i,j,t=0,imin;
   for(i=0;i<n-1;i++)
{
     imin=i;
     for(j=i+1;j<n;j++)
         if(b[j]<b[imin])
             imin=j;
     t=b[i];b[i]=b[imin];b[imin]=t;
   }
}
void main()
{
   int a[10]={5,67,3,89,8,1,7,23,9,12};
   int i;
```

```
    for(i=0;i<10;i++)
    printf("%4d",a[i]);        /*排序前输出 a 数组元素值*/
    printf("\n");
    sort(a,10);
    for(i=0;i<10;i++)
        printf("%4d",a[i]);  /*排序后输出 a 数组元素值*/
    printf("\n");
}
```

运行结果为：

```
5  67   3  89   8   1   7   2  39  12
1   3   5   7   8   9  12  23  67  89
```

分析：由运行结果可见，排序前 a 数组是按原来的赋值顺序存储的，排序后 a 数组是按升序顺序存储的，如图 7-3 所示，实参 a 数组与形参 b 数组共享同一段内存空间，对 b 数组进行排序，也就是对 a 数组排序。需要注意的是，b 数组的存在是暂时的，调用 sort 函数时 b 数组才开始存在，调用 sort 函数结束后，b 数组就不存在了。

排序前 a 数组存储情况

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	67	3	89	8	1	7	23	9	12
b[0]	b[1]	b[2]	a[3]	ba[4]	b[5]	b[6]	b[7]	b[8]	b[9]

排序后 a 数组存储情况

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	3	5	7	8	9	12	23	67	89
b[0]	b[1]	b[2]	a[3]	ba[4]	b[5]	b[6]	b[7]	b[8]	b[9]

图 7-3　排序前、后 a 数组存储情况

关于数组名作函数参数的说明：

（1）用数组名作函数参数，必须在主调函数和被调函数中分别定义实参数组和形参数组。

（2）形参数组和相对应的实参数组类型、位置必须一一对应，二者不一致时，即会发生错误。

（3）定义形参数组时，可以说明数组的大小，也可以不说明数组的大小，但空方括号不能省略。若需知道数组的大小，可另设一个参数，传递需处理的数组元素个数，如例 7.11 所示。

（4）由前面的分析可知，数组名作函数参数时，若形参数组的数组值发生变化，则实参数组的数组值也要发生相同变化。由这点看，似乎参数的传递是双向的，但这是错的。在 C 语言中，无论是数值传递还是地址传递都是是单向的，只能是由实参数组传递给形参数组，不能反向传递。

（5）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，虽不致于出现语法错误（编译能通过），但程序的运行结果将与实际不符，这是应予以注意的。

【例 7.12】形参数组和实参数组长度不相同的问题。

```
#include <stdio.h>
void p(int a[8])
{  int i;
   printf("\nvalues of array a are:\n");
   for(i=0;i<8;i++)
   printf("%3d",a[i]);
}
void 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("%3d",b[i]);
   printf("\n");
   p(b);
   printf("\nlast values of array b are:\n");
   for(i=0;i<5;i++)
   printf("%3d",b[i]);
   printf("\n");
}
```

程序运行过程与结果为：

```
input 5 numbers:
1↙
3↙
5↙
7↙
9↙
initial values of array b are:
1  3  5  7  9
values of array a are:
1    3    5    7    9  1245120  4199417   1
last values of array b are:
1  3  5  7  9
```

分析：在本例中，形参数组长度与实参数组长度不同，实参数组长度为 5，形参数组长度改为 8。虽然形参数组 a 和实参数组 b 的长度不一致，但编译能够通过。从结果看，数组 a 的元素 a[5]、a[6]、a[7]显然是无意义的。

字符串可以理解为字符数组，字符数组名也可以作为函数的参数。其详细操作见指针一章。

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。以下写法都是合法的。

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

或

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

【例 7.13】在一个 3×3 的二维数组中，求所有元素中的最大值。

```
#include <stdio.h>
int Max (int array[ ][3])
{   int i, j, k, max;
    max=array[0][0];
    for (i=0; i<3; i++)
       for (j=0; j<3; j++)
         if (array[i][j]>max) max=array[i][j];
    return max;
 }
 void main()
 {   int a[3][3]={{8,3,5}, {1,4,6}, {15, 17, 34}};
     printf("\n max value is %d\n", Max(a));
 }
```

运行结果为：

```
max value is 34
```

注意：max 与 Max 不同，max 是变量名，而 Max 是函数名。

7.6 函数的嵌套调用和递归调用

前面介绍过函数之间允许相互调用，也允许嵌套调用，函数还可以自己调用自己，称为递归调用。

7.6.1 函数的嵌套调用

C 语言中不允许函数嵌套定义。因此各函数之间是平行的，不存在上一级函数和下一级函数的问题。但是C 语言允许函数的嵌套调用。即在调用一个函数的过程中，又调用另一个函数，换句话说，在被调函数中又调用其他函数。这与其他语言的子程序嵌套的情形是类似的。其关系如图 7-4 所示。

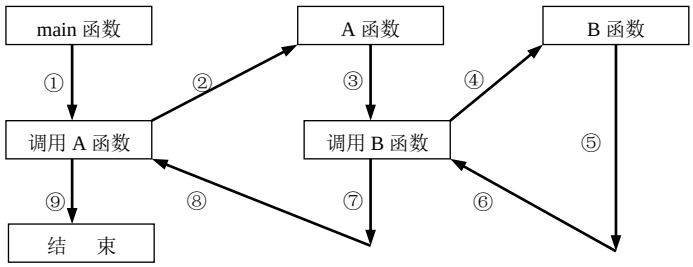


图 7-4　函数的嵌套调用

图 7-4 表示了两层嵌套的情形。其执行过程是：执行 main 函数中调用 a 函数的语句时，即转去执行 a 函数，在 a 函数中调用 b 函数时，又转去执行 b 函数，b 函数执行完毕返回 a 函数的断点继续执行，a 函数执行完毕返回 main 函数的断点继续执行。

【例 7.14】计算 $2^2!+3^2!$的阶乘和 s。

```
#include <stdio.h>
```

```
long f1(int p)          /*函数 f1 为计算平方值*/
{  int k;
   long r;
    long f2(int);       /*对函数 f2 的引用说明*/
    k=p*p;
    r=f2(k);            /*调用函数 f2 计算阶乘值*/
    return r;           /*将阶乘值返回函数 main*/
}
long f2(int q)          /*函数 f2 为计算阶乘值*/
{  long c=1;
   int i;
   for(i=1;i<=q;i++)
   c=c*i;
   return c;            /*将阶乘值返回函数 f1*/
}
main()
{  int i;
   long s=0;
   for (i=2;i<=3;i++)   /*两次调用函数 f1，实参分别为 2 和 3*
   s=s+f1(i);           /*两次调用函数 f1 的返回值相加求和，即为 2²!+3²!的和 s */
   printf("\ns=%ld\n",s);
}
```

运行结果为：

```
S=362904
```

分析：本题编写了两个函数，一个是用来计算平方值的函数 f1，另一个是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。函数调用的顺序如图 7-5 所示。

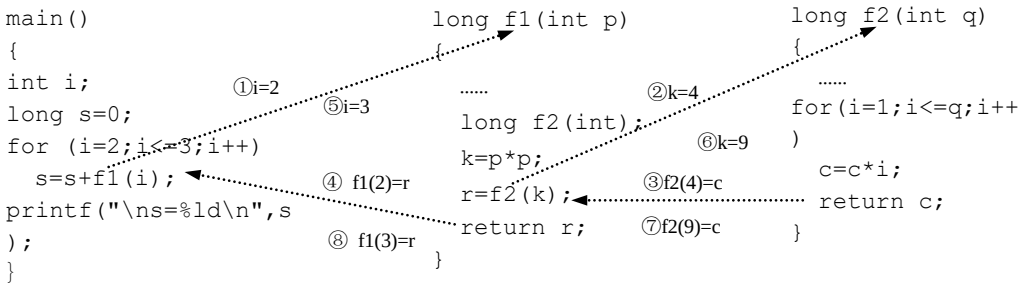


图 7-5　函数调用的顺序

7.6.2　函数的递归调用

在调用一个函数的过程中又直接或间接地调用函数本身称为递归调用，这种函数称为递归函数。递归调用分两种方式，一种是直接递归调用，另一种是间接递归调用。C 语言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直接或间接

地调用其自身。每调用一次就进入新的一层。

例如直接递归调用方式，有函数 f 如下：

```
int f(int x)
{
   int y;
   z=f(y);
   return z;
}
```

例如，间接递归调用方式，有函数 f1 和 f2 如下：

```
int f1(x)
int x;
{  int y,z;
   ……
   z=f2( y);
   ……
   return (2*z);
}
int f2(t)
int t;
{  int a,c;
   ……
   c=f1(a);
   ……
   return (3+c);
}
```

由上述两个程序可以看出，无论是直接递归调用，还是间接递归调用，函数将无休止地直接或间接调用其自身，如图 7-6 和 7-7 所示，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。下面举例说明递归调用的执行过程。

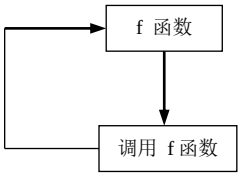


图 7-6　直接递归调用

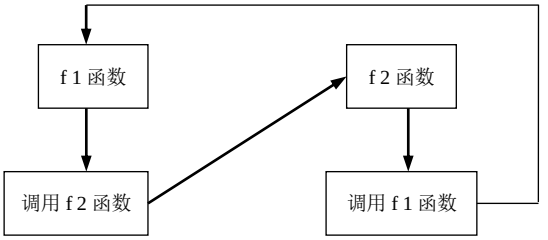


图 7-7　间接递归调用

【例 7.15】用递归法计算 *n*!。

可用下述公式表示：

$n!=1$ ($n=0,1$)

$n!=n\times(n-1)!$ ($n>1$)

按公式可编程如下：

```
#include <stdio.h>
long fact(int n)
{  long f;
   if(n<0) printf("n<0,input error");  /* n<0,无结果*/
   else if(n==0||n==1) f=1;            /*n==0 或 n==1 时, n! =1*/
   else f=fact(n-1)*n;                 /*递归调用fact函数,fact(n)=n*fact(n-1)*/
   return(f);
}
main()
{  int n;
   long y;
   printf("\ninput a inteager number:\n");
   scanf("%d",&n);
   y=fact(n);
   printf("%d!=%ld",n,y);
}
```

分析：函数的递归调用有两个过程，一是下推，一是回代。在本例中函数 fact 是一个递归函数。主函数调用 fact 后，即进入函数 fact 内部执行，当 n<0、n=0 或 n=1 时，都将结束函数的执行，否则就递归调用 fact 函数自身。下面以 n=9 为例进行 fact 函数自身调用说明：fact(9)调用 fact(8)（fact(9)=9*fact(8)），fact(8)调用 fact(7)（fact(8) = 8*fact(7)），…，fact(2)调用 fact(1)（fact(2)=2*fact(1)），当 fact 函数的实参值为 1 时，n!=1 递归终止，这个过程叫做“下推”。然后可逐层退回，fact(1)=1，fact(2)=2*1，…，fact(9)=9*8*7*6*5*4*3*2*1，这个过程叫作“回代”。回代结束后，将 fact(9)的值返回给主函数 main()。函数 fact(9)的下推和回代过程如图 7-8 所示。

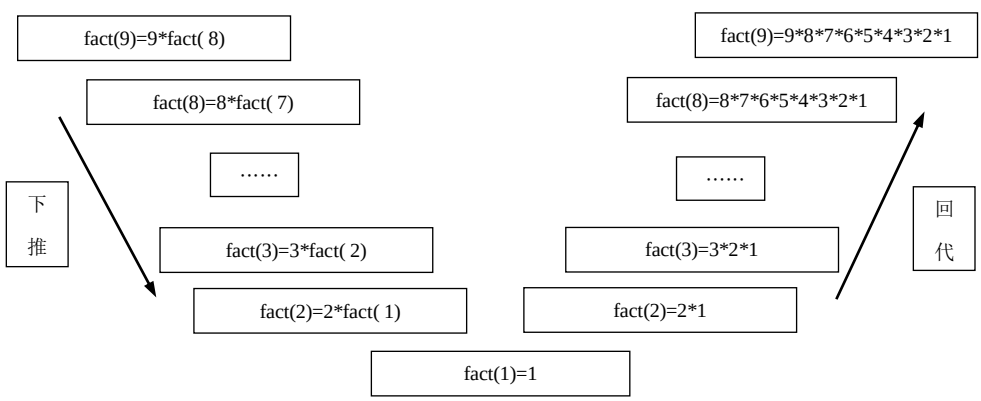


图 7-8　函数 fact(9)的下推和回代过程

7.7 变量的作用域与存储类别

在讨论函数的参数传递时曾经提到，形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。原理是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。这一点表明形参变量的作用范围只在被调函数内才是有效的，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。C 语言中所有的变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。

7.7.1 局部变量

局部变量也称为内部变量，是在函数内部定义的变量。其作用域仅限于函数内，在该函数以外使用这种变量，就是非法的。

例如：

```
int f1(int a)        /*函数 f1*/
{   int b,c;
    ……                  } a，b，c 的作用范围
}
int f2(int x)        /*函数 f2*/
{   int y,z;
    ……                  } x，y，z 的作用范围
}
main()
{   int k,n;
    ……                  } k，n 的作用范围
}
```

在函数 f1 内定义了 3 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。k、n 的作用域限于 main 函数内。

关于局部变量作用域的说明：

（1）主函数中定义的变量也是局部变量，只能在主函数中使用，不能在其他函数中使用。同理，其它函数中定义的变量在主函数中也不能使用。因为主函数也是一个函数，它与其他函数是平行关系，互不包含。

（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）在不同的函数中允许使用相同的变量名，因为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。如在例 7.15 中，形参和实参的变量名都为 n，是完全允许的。

（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,m;
     ……
    {   int n;
        ……          } n 作用域
    }                  } s，m 的作用域
    ……
}
```

【例 7.16】关于局部变量的例题。

```
#include <stdio.h>
void main()
{    int k=5;
   {  int k=8;
      printf("k=%d\n",k);
    }
    printf("k=%d\n",k);
}
```

运行结果为：

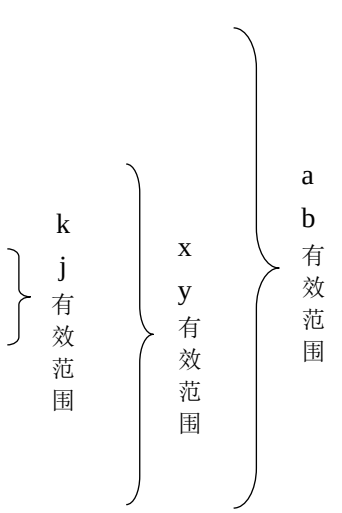
```
k=8
k=5
```

分析：本题在 main 中定义了两个变量 k，其中一个在复合语句内，一个在复合语句外。应该注意这两个 k 不是同一个变量。在复合语句外由在复合语句外定义的 k 起作用（k=5），而在复合语句内则由在复合语句内定义的 k 起作用（k=8）。因此程序先输出在复合语句内定义的 k 值（k=5），后输出在复合语句外定义的 k 值（k=8）。

7.7.2　全局变量

全局变量也称为外部变量，它是在函数外部定义的变量。它不属于哪一个函数，它属于一个源程序文件。其作用范围是在其所定义点之后一直到本程序结束的程序部分中有效。在函数中使用全局变量，一般应作全局变量说明。只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。但在一个函数之前定义的全局变量，在该函数内使用可不再加以说明。例如：

```
int a,b;              /*全局变量*/
void f1()             /*函数 f1*/
{
    ……
}
float x,y;            /*全局变量*/
int f2()              /*函数 fz*/
{
    extern int k,j; /*说明全局变量 k,j */
    ……
}
main()                /*主函数*/
{
    ……
```



```
}
int k,j;
```

从上面程序可以看出，a、b、x、y、k、j 都是在函数外部定义的全局变量，都是全局变量。a、b 定义在程序最前面，因此在 f1、f2 及 main 内不加说明即可使用；x、y 定义在函数 f2 之前，因此在 f2 和 main 内不加说明即可使用；k、j 定义在程序最后面，因此在 f1、f2 及 main 内不加说明是不能使用的。但在 f2 中对 k、j 进行了说明，因此在 f2 中就可以引用变量 k 和 j 了。

由前面的知识可以知道，函数间的参数传递是单向的，只能有一个返回值。若想利用参数传递返回多个值，是不可能的，但利用全局变量可以做到。这时函数可以返回多个值。

【例 7.17】利用全局变量，编程求半径为 r 的圆的周长和面积以及半径为 r 的球体的体积。

```
#include <stdio.h>
#define Pi 3.1415926
float S,V;
float f(float r)
{   float L;
    L=2*Pi*r;
    S=Pi*r*r;
    V=4.0/3*Pi*r*r*r;
    return(L);
}
main()
{   float r,L;
    printf("\ninput a number r:\n");
    scanf("%f",&r);
    L=f(r);
    printf("L=%6.2f,S=%6.2f,V=%6.2f\n",L,S,V);
}
```

运行过程与结果为：

```
input a number r:
4
L= 25.13,S= 50.27,V=268.08
```

分析：本例中定义了两个全局变量 S、V，用来存放圆的面积和球的体积，其作用域为整个程序。函数 f 的返回值为圆的周长 L。由主函数完成半径的输入及结果输出。由于函数只有一个返回值，而本题需要 3 个返回值，因此用全局变量 S、V 来补充 2 个返回值。全局变量是实现函数之间数据通信的有效手段。

关于全局变量的说明：

（1）若在一个函数之前定义了全局变量，在该函数内使用可不加说明；若全局变量的定义是在所使用该变量的函数下方，则必须在这个函数中对该全局变量进行说明，且在整个程序内，可对全局变量进行多次说明。

全局变量说明的一般形式为：

```
extern 类型说明符 变量名，变量名，…;
```

全局变量在定义时就已分配了内存单元，因此全局变量定义时可赋初值，但说明全局变量时是不能再赋初值，因为变量的说明只是表明在函数内要使用某全局变量。

（2）全局变量可加强函数模块之间的数据联系，若全局变量在其中一个函数中发生变化，则在其他函数中该全局变量要同时发生变化。这样就使函数要依赖这些变量，因而使得函数的独立性降低。从模块化程序设计的观点来看这是不利的，因此在不必要时尽量不要使用全局变量。

（3）在同一源文件中，允许全局变量和局部变量同名。在局部变量的作用域内，全局变量不起作用。

【例 7.18】全局变量和局部变量同名。

```
#include <stdio.h>
int f(int a,int b)
{  extern int h;        /*说明全局变量 h */
   int v;
   v=a*b*h;
   return v;
}
main()
{  extern int b,h;      /*说明全局变量 b,h */
   int a=5;
   printf("v=%d",f(a,b));
}
int a=3,b=4,h=5;        /*定义全局变量 a,b,h*/
```

运行结果为：

```
v=100
```

分析：在本例的程序中，全局变量在最后定义，因此在前面函数中必须对要用的全局变量进行说明。全局变量 a、b 和 f 函数的形参 a、b 同名。全局变量都赋了初始值，main 函数中也对局部变量 a 赋了初始值。执行程序时，在 printf 语句中调用 f 函数，实参 a 的值应为 main 中定义的局部变量 a 的值，等于 5，全局变量 a 在 main 内不起作用；实参 b 的值为全局变量 b 的值为 4，进入 f 后这两个值传送给形参 a 和 b，f 函数中使用的 h 为全局变量，其值为 5，因此 v 的计算结果为 100，返回主函数后输出。

7.7.3　变量的存储类型

各种变量的作用域不同，就其本质来说是因变量的存储类型不同。所谓存储类型是指变量占用内存空间的方式，也称为存储方式。

变量的存储方式可分为“静态存储”和“动态存储”两种。

静态存储变量通常是在变量定义时就分配存储单元并一直保持不变，直至整个程序结束。7.7.2 节中介绍的外部变量即属于此类存储方式。动态存储变量是在程序执行过程中，使用它时才分配存储单元，使用完毕立即释放。典型的例子是函数的形式参数，在函数定义时并不给形参分配存储单元，只是在函数被调用时，才予以分配，调用函数完毕立即释放。如果一个函数被多次调用，则反复地分配、释放形参变量的存储单元。

从以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在，时而消失。把这种由于变量存储方式不同而产生的特性称为变量的生存期。生存期表示了变量存在的时间。生存期和作用域是从时间和空间这两个不同的角度来描述变量的特性，这两者既有联系，

又有区别。一个变量究竟属于哪一种存储方式，并不能仅从其作用域来判断，还应有明确的存储类型说明。

在 C 语言中，对变量的存储类型说明有以下 4 种。

auto：自动变量

register：寄存器变量

extern：外部变量

static：静态变量

自动变量和寄存器变量属于动态存储方式，外部变量和静态变量属于静态存储方式。在介绍了变量的存储类型之后，可以知道对一个变量的定义不仅应说明其数据类型，还应说明其存储类型。因此变量定义的完整形式应为：

```
存储类型说明符 数据类型说明符 变量名，变量名…;
```

例如：

```
static int a,b;                  定义 a,b 为静态类型变量
auto char c1,c2;                 定义 c1,c2 为自动字符变量
static int x[5]={1,2,3,4,5};     定义 x 为静态整型数组
extern int x,y;                  定义 x,y 为外部整型变量
```

下面分别介绍以上 4 种存储类型。

1．auto 型自动变量

这种存储类型是 C 语言程序中使用最广泛的一种类型。C 语言规定，函数内凡未加存储类型说明的局部变量均视为自动变量，也就是说自动变量可省去说明符 auto。在前面各章的程序中所定义的局部变量凡未加存储类型说明符的都是自动变量。例如：

```
{   int x,y,z;
    char c;
    ……
}
```

等价于：

```
{   auto int x,y,z;
    auto char c;
    ……
}
```

自动变量具有以下特点：

（1）自动变量的作用域与局部变量相同，仅限于定义该变量的个体内。在函数中定义的自动变量只在该函数内有效。在复合语句中定义的自动变量只在该复合语句中有效。

（2）自动变量属于动态存储方式，只有在使用它，即定义该变量的函数被调用时才给它分配存储单元，开始它的生存期。函数调用结束，释放存储单元，结束生存期。因此函数调用结束之后，自动变量的值不能保留。在复合语句中定义的自动变量，在退出复合语句后也不能再使用，否则将引起错误。

（3）由于自动变量的作用域和生存期都局限于定义它的个体内（函数或复合语句内），因此不同的个体中允许使用同名的变量而不会混淆。即使在函数内定义的自动变量也可与该函数内部的复合语句中定义的自动变量同名。

2．extern 型外部变量

在全局变量部分已经介绍过全局变量就是外部变量，这是对同一类变量的两种不同角度的提法。全局变量是从它的作用域提出的，外部变量是从它的存储方式提出的，表示了它的生存期。外部变量不仅可以在同一个文件的不同函数中应用，还可以在多个文件中应用。

例如，有一个程序由文件 F1.cpp 和 F2.cpp 组成：

```
/*F1.cpp 文件*/
#include <stdio.h>
#include "F2.cpp"            /*包含文件 F2.cpp */
int x,y;                  /*外部变量定义*/
char z;                   /*外部变量定义*/
main()
{  extern func();         /*说明外部函数*/
   x=5;y=9;z='k';
   func();
}
/*F2.cpp 文件*/
#include <stdio.h>
extern int x,y;           /*外部变量说明*/
extern char z;            /*外部变量说明*/
func()
{   printf("x=%d,y=%d,z=%c",x,y,z);
}
```

运行结果为：

```
x=5,y=9,z=k
```

分析：在 F1.cpp 和 F2.cpp 两个文件中都要使用 x、y、z 这 3 个变量。在 F1.c 文件中把 x、y、z 都定义为外部变量，并且赋了值，然后调用了 F2.cpp 文件中的函数 func()。在 F2.cpp 文件中用 extern 把 3 个变量说明为外部变量，表示这些变量已在其他文件中定义了，编译系统不再为它们分配内存空间。由结果可以知道，在 F1.cpp 中给变量 x、y、z 赋值，在 F2.cpp 中输出了变量 x、y、z 的值，说明了两个文件中是同一组 x、y、z 变量。若 x、y、z 在 F1.cpp 中改变，则 F2.cpp 中的 x、y、z 的值会同时发生变化，因此要慎重使用外部变量。

对子构造类型的外部变量，如数组等可以在说明时赋初值，若不赋初值，则系统会自动定义它们的初值为 0。

3．static 型静态变量

静态变量属于静态存储方式，但是属于静态存储方式的量不一定就是静态变量，例如外部变量虽属于静态存储方式，但不一定是静态变量。只有用 static 定义的变量才是静态变量。外部变量用 static 定义就是静态外部变量。对于自动变量，虽然它属于动态存储方式，但是也可以用 static 定义它为静态自动变量，或称静态局部变量，从而成为静态存储方式。

由此看来，一个变量可由 static 进行再说明，并改变其原有的存储方式。

（1）静态局部变量

静态局部变量的特点：

① 从生存期来看，静态局部变量的生存期为整个源程序。它在函数内定义，但不像局部变量那样，函数执行时就存在，函数结束执行时就消失，静态局部变量始终存在，除非结束整个程序的运行。

② 从作用域来看，静态局部变量的作用域仍与自动变量相同，即只能在定义该变量的函数内使用该变量。退出该函数后，尽管该变量还继续存在，但不能使用它。

③ 允许对构造类静态局部量赋初值。若未赋以初值，则由系统自动赋以 0 值。

④ 对基本类型的静态局部变量若在说明时未赋以初值，则系统自动赋予 0 值。而对自动变量不赋初值，则其值是不定的。

根据静态局部变量的特点，可以看出它是一种生存期为整个源程序的变量。虽然离开定义它的函数后不能使用，但如再次调用定义它的函数时，它又可继续使用，而且保存了前次被调用后留下的值。因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可考虑采用静态局部变量。虽然用全局变量也可以达到上述目的，但全局变量有时会造成意外的副作用，因此仍以采用局部静态变量为宜。

【例 7.19】静态局部变量。

```
#include <stdio.h>
void f()
{   static int j=10;
    printf("j=%d\n",j);
    j=j+10;
}
main()
{   f();
    f();
    f();
}
```

运行结果为：

```
j=10
j=20
j=30
```

分析：函数 f 中定义了一个静态变量 j，其初值为 10。当第 1 次调用函数 f 时，输出 j=10，然后 j 的值加 10，成为 20。因为 j 是静态变量，所以 j 的值（20）会保存在内存中，直到函数 f 执行完毕；当第 2 次调用函数 f 时，j 中的值是第 1 次调用函数 f 后所保留的 20，而不重新设初值，因此输出 j=20，然后 j 的值加 10，成为 30；当第 3 次调用函数 f 时，输出 j=30。

（2）静态全局变量

全局变量与静态全局变量都是静态存储方式。这两者在存储方式上并无不同，其不同在于作用域上。当一个程序由多个文件组成时，非静态的全局变量在各个文件中都是有效的；而静态全局变量则限制了其作用域，即只在定义该变量的文件内有效，在同一程序的其他文件中不能使用它。由于静态全局变量的作用域局限于一个文件内，只能为该文件内的函数公用，

因此可以避免在其他文件中引起错误。

4．register 型寄存器变量

上述各类变量都存放在内存储器中，当对一个变量频繁存取时，必须要反复访问内存储器，从而花费大量的存取时间。为此，C 语言提供了另一种变量，即寄存器变量。这种变量存放在 CPU 的寄存器中，使用时，不需要访问内存，而直接从寄存器中存取，这样可提高效率。一般对于循环次数较多的循环控制变量及循环体内反复使用的变量均可定义为寄存器变量。

【例 7.20】求 1^2～100^2 的平方和。

```
#include <stdio.h>
main()
{   register i;
    register long s=0;
    for(i=1;i<=100;i++)
    s=s+i*i;
    printf("s=%ld\n",s);
}
```

运行结果为：

```
S=338350
```

本程序循环 100 次，i 和 s 都将频繁使用，因此可定义为寄存器变量。

关于寄存器变量的说明：

（1）因为寄存器变量属于动态存储方式。只有局部变量和形式参数才可以定义为寄存器变量。静态存储方式的变量不能定义为寄存器变量。

（2）由于计算机 CPU 中寄存器的个数是有限的，因此使用寄存器变量的个数也是有限的。

（3）在 Visual C++中，实际上是把寄存器变量当成自动变量处理的，因此速度并不能提高。而在程序中允许使用寄存器变量只是为了与标准 C 保持一致。

7.7.4　函数的作用域

由上面介绍可知，不同的变量有不同的作用域，事实上函数也有作用域。C 语言按作用域不同将函数分为两类：内部函数和外部函数。

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。定义内部函数的一般形式如下。

格式：

```
 static 类型说明符 函数名(形参表)
```

例如：`static int f(int a,int b)`

内部函数也称为静态函数。但此处静态 static 的含义已不是指存储方式，而是指对函数的调用范围只局限于本文件，因此在不同的源文件中定义同名的静态函数不会引起混淆。

2．外部函数

外部函数在整个源程序中都有效，其定义的一般形式如下。

格式：

```
extern 类型说明符 函数名(形参表)
```

例如：`extern int f(int a,int b)`

如在函数定义中没有说明 extern 或 static，则隐含为 extern。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，应用 extern 说明被调函数为外部函数。例如：

```
Func1.cpp (源文件 1)
main()
{   extern int f1(int x);    /*外部函数说明，表示 f1 函数在其他源文件中*/
    ……
}
Func2.cpp (源文件 2)
extern int f1(int x);        /*外部函数定义*/
{  …… }
```

7.8 编译预处理

在 C 语言中，以“#”号开头的命令为预处理命令。利用预处理命令可以进行文件包含、宏定义、条件编译等 C 语言的环境设置，进而提高编程效率。但预处理命令不是 C 语言本身的组成部分，而是 ANSI C 统一规定的，C 语言编译系统不识别预处理命令，不能对预处理命令进行直接编译，因此在程序编译之前，必须对这些特殊的命令进行“预处理”，使之在运行时可以执行。然后才可以对程序的其他部分进行编译，所以把这个过程叫做“编译预处理”。

由于预处理命令不是 C 语言本身的组成部分，因此书写程序时预处理命令应写在函数的外部，通常是源程序的开始部分，且命令后不加“;”。

7.8.1 文件包含

文件包含是编译预处理的一个重要功能。通过文件包含可以调用各种库函数及多个源程序。文件包含命令行的一般形式为：

格式：

```
#include "文件名"
```

或 `#include <文件名>`

例如：`#include"stdio.h"`、`#include"math.h"`

文件包含命令的功能是把指定的文件插入该命令行位置取代该命令行，从而把指定的文件和当前的源程序文件连成一个源文件。

关于文件包含命令的说明：

（1）<文件名>：使用尖括号表示在包含文件目录中去查找（包含目录是由用户在设置环境时设置的），而不在源文件目录中去查找。

"文件名"：使用双引号则表示首先在当前的源文件（用户编写的.cpp 文件）目录中查找，若未找到才到包含目录中去查找。用户编程时可根据自己文件所在的目录来选择某一种命令形式。文件名应是文件的全名，即文件名及扩展名。

（2）一个包含命令只能指定一个被包含文件，若有多个文件要包含，则需用多个包含命令。

（3）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。

C 语言是结构化程序设计语言。在程序设计中，文件包含是很有用的。一个大的程序可以分为多个模块（函数），由多个程序员分别编程。最后将几个文件（模块）通过文件包含将之合为一个文件。

【例 7.21】多文件调用。由键盘输入 10 个数，找出其最大值与最小值并计算出平均值。

```
/*func7.21.cpp*/
#include <stdio.h>
#include "7.211.cpp"                          /*包含文件 7.211.cpp */
#include "7.212.cpp"                          /*包含文件 7.212.cpp */
#include "7.213.cpp"                          /*包含文件 7.213.cpp */
main()
{   extern int mmax(int a[],int n);     /*声明外部函数 mmax */
    extern int mmin(int a[],int n);     /*声明外部函数 mmin */
    extern float avg(int a[],int n);    /*声明外部函数 avg */
    float average;
    int i,max,min,b[10];
    for (i=0;i<10;i++)
         scanf("%d",&b[i]);
    max=mmax(b,10);                     /*调用外部函数 mmax */
    min=mmin(b,10);                     /*调用外部函数 mmin */
    average=avg(b,10);                  /*调用外部函数 avg */
    printf("max=%d,min=%d,average=%.2f",max,min,average);
}
/*func7.211.cpp*/
#include <stdio.h>
int mmax(int a[],int n)                 /*定义外部函数 mmax */
{   int i;
    int max=a[0];
    for (i=1;i<n;i++)
    if (a[i]>max)
        max=a[i];
    return(max);
}
/*func7.212. cpp*/
#include <stdio.h>
int mmin(int a[],int n)                 /*定义外部函数 mmin */
{   int i;
    int min=a[0];
    for (i=1;i<n;i++)
    if (a[i]<min)
        min=a[i];
return(min);
}
/*func7.213. cpp*/
#include <stdio.h>
float avg(int a[],int n)                /*定义外部函数 avg*/
{   int i;
    float aver,sum=0;
    for (i=0;i<n;i++)
    sum=sum+a[i];
```

```
    aver=sum/n;
    return(aver);
}
```

运行过程与结果为：

```
0↙
1↙
2↙
3↙
4↙
5↙
6↙
7↙
8↙
9↙
max=9,min=0, average=4.50
```

分析：本例中用到了 4 个文件，其中 func7.21 为主函数，其他 3 个为计算最大值、最小值及平均值的文件。在文件 func7.21 中用#include "7.211.cpp"，#include "7.212.cpp"，#include "7.213.cpp"三条包含语句，将 4 个文件连结成 1 个文件，即相当于文件"7.211.cpp"，"7.212.cpp"，"7.213.cpp"都在文件"7.21.cpp"中。

7.8.2 宏定义

宏定义是指用一个宏名（即名字）来代表一个字符串。宏定义的一般形式如下。

格式：

```
#define 宏名[(形参表)] 字符串
```

宏定义的功能是在编译预处理时，对程序中所有出现的“宏名”都用宏定义中的字符串去代换，这称为“宏代换”或“宏展开”。

例如：

```
#define Pi 3.1415926
#define S(Pi*R*R)
```

说明：

（1）“define”为宏定义命令。

（2）“宏名”为所定义的宏的名字，一般用大写字符表示。宏名后省略圆括号为“无参数宏定义”，未省略圆括号为“有参数宏定义”。

（3）宏定义时的参数表为形式参数表，参数之间用“，”隔开。在宏调用中的参数称为实际参数。对于带参数的宏，在调用中，不仅要将宏展开，而且要用实参去代换形参。带参宏调用的一般形式如下。

格式：

```
宏名(实参表);
```

（4）“字符串”可以是常数、表达式、格式串等。

（5）宏定义允许嵌套，在宏定义的字符串中可以使用已经定义的宏名。在宏展开时由预处理程序层层代换，见例 7.22。但预处理程序对宏代换不做任何检查。如有错误，只能在编译已被宏展开后的源程序时才会发现。

（6）宏定义必须写在函数之外，其作用域为从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，例如：

```
 # define PI 3.14159
main()
{          ┐
……         ├ PI 的作用域
}          ┘
# undef PI
f1()
{
.... /*表示 PI 只在 main 函数中有效，在 f1 中无效*/
}
```

（7）可用宏定义表示数据类型，使书写方便。

例如： `#define STU struct stu(结构体类型)`

在程序中可用 STU 作变量说明：STU body[5],*p;。由于宏代换是用“字符串”简单替换宏名，因此在使用时要分外小心，以免出错。

在程序设计中，有些常量或表达式需要多次应用，可以把它们定义为符号常量或宏，当需要变化时，可避免多次修改，从而节省时间，并减少出错。

下面介绍宏定义的应用。

1．无参数宏定义的应用

无参数宏经常用于比较简单的宏代换。

【例 7.22】计算 3!+5!+7!的和。

```
#include <stdio.h>
#define M 1*2*3
#define N M*4*5
#define Q N*6*7
void main()
{ int S=0;
  S=M+N+Q;
  printf("3!+5!+7!=%d\n",S);
}
```

运行结果为：

```
3!+5!+7!=5166
```

分析：在程序中，宏代换就是用“字符串”简单替换宏名。例如，表达式 S=M+N+Q 替换后为 S=1*2*3+1*2*3*4*5+1*2*3*4*5*6*7=5166。

对于具有多种运算符的字符串在进行宏代换时，其加圆括号与不加有本质的区别。看下面的例题。

【例 7.23】计算 $4(4x+y^2)+5(x^2-5y)$。

```
/*a 程序（字符串不加圆括号）*/
#include <stdio.h>
#define M 4*x+y*y
#define N x*x-5*y
void main()
```

a 程序（字符串不加圆括号）的运行过程与结果：

```
2↙
3↙
z=46.00
```

```
{ float x,y,z;
  scanf("%f",&x);
  scanf("%f",&y);
  z=4*M+5*N;
  printf("z=%.2f\n",z);
}
/*b 程序（字符串加圆括号）*/
#include <stdio.h>
#define M (4*x+y*y)
#define N (x*x-5*y)
void main()
{ float x,y,z;
  scanf("%f",&x);
  scanf("%f",&y);
  z=4*M+5*N;
  printf("z=%.2f\n",z);
}
```

b 程序（字符串加圆括号）的运行过程与结果：

```
2↙
3↙
z=13.00
```

分析：由两个程序的运行结果可以看出，在由键盘输入 x 和 y 的值相同的情况下，运行结果是不同的。区别在于宏代换只是简单地用字符串替换宏名，字符串保持不变，因此字符串加与不加圆括号，运行结果当然就不同了。下面是两个程序的宏代换展开过程：

a 程序：`z=4*M+5*N=4*4*x+y*y+5*x*x-5*y=4*4*2+3*3+5*2*2-5*3=32+9+20-15=46`

b 程序：`z=4*M+5*N=4*(4*x+y*y)+5*(x*x-5*y)`

`=4*(4*2+3*3)+5*(2*2-5*3)=4*17-5*11=68-55=13`

对“输出格式”进行宏定义，可以减少书写麻烦。

【例 7.24】对“输出格式”进行宏定义。

```
#include <stdio.h>
#define P printf
#define D "%d\n"
#define F "%.2f\n"
#define S "%s\n"
main()
{  int x=5, y=8;
   float a=3.8,b=9.7;
   char str1[]="China",str2[]="Beijing";
   P(D F,x,a);
   P(D S,y,str1);
   P(F S,b,str2);
}
```

运行结果为：

```
5
3. 80
8
China
9.70
Beijing
```

本题中可以将“输出格式”编辑成头文件，只要将该头文件包含进源程序中，就可应用头文件中的“输出格式”。

建头文件的过程与 C 源程序相同，只是保存时将扩展名设为.h 即可。

```
/*头文件 format.h*/
#define P printf
#define D "%d\n"
#define F "%.2f\n"
#define S "%s\n"
```

例 7.24 可改为：

```
#include <stdio.h>
#include "format.h"
main()
{ int x=5, y=8;
  float a=3.8,b=9.7;
  char str1[]="China",str2[]="Beijing";
  P(D F,x,a);
  P(D S,y,str1);
  P(F S,b,str2);
}
```

同学们可以验证一下，运行结果是否和例 7.24 相同。

2．有参数宏定义的应用

对于带参数的宏，在调用中，不仅要宏展开，而且要用实参去代换形参。

例如：

```
#define Y(x) x*x-3*x+4      /*宏定义*/
k=Y(5);                     /*宏调用*/
```

在进行宏调用时，用实参 5 去代替形参 x，经预处理宏展开后的语句为：

```
 k=5*5-3*5+4
```

关于带参数宏定义的说明：

（1）在带参宏定义中，宏名和形参表之间不能有空格出现，若有空格，将被认为是无参宏定义,会将空格后的字符都当作替换字符串的一部分。例如：若 Y 与（x）之间有空格：

```
#define Y (x) x*x-3*x+4     /*宏定义*/
k=Y(5);                     /*宏调用*/
```

则 k=Y(5)的宏展开后的语句为：

```
k=(x) x*x-3*x+4(5)
```

显然是错误的。

（2）带参宏的调用，有实参向形参的数据传递，但这种传递与函数参数传递是不同的。在函数中，形参和实参是两个不同的量，各有自己的作用域，调用时要把实参值赋予形参，进行“值传递”。而在带参宏中，只是字符串简单代换，不存在值传递的问题。在带参宏定义中，形式参数不分配内存单元，因此不必进行类型定义。而宏调用中的实参有具体的值。要用它们去代换形参，因此必须进行类型说明。

（3）在宏定义中，字符串内的形参通常要用括号括起来以避免出错，见下面例题。

【例 7.25】计算 $2^2+4^2+6^2$ 的和。

```
/*7.25a*/
#include <stdio.h>
```

```
/*7.25b*/
#include <stdio.h>
#define S(x) x*x
main()
{
   int a=2,sum;
   sum=S(a)+S(a+2)+S(a+4);
```

```
#define S(x) (x)*(x)
main()
{
   int a=2,sum;
   sum=S(a)+S(a+2)+S(a+4);
   printf("sum=%d\n",sum);
}
```

运行结果为：

```
sum=56
```

分析：由于两个程序的宏展开不同，其结果也不同，7.25a 程序结果正确，7.25b 程序结果是错的。看一下两个程序的宏展开，就可明白区别。

```
7.25a  sum=S(a)+S(a+2)+S(a+4)=a*a+(a+2)*(a+2)+(a+4)*(a+4)
7.25b  sum=S(a)+S(a+2)+S(a+4)=a*a+a+2*a+2+a+4*a+4
```

（4）调用函数只能返回一个值，利用宏可以得到多个值，即宏定义可定义多个语句。

【例 7.26】计算 x 的平方根和平方，求 x、y 两个数中的最小值。

```
#include <stdio.h>
#include <math.h>
#define S(S1,S2,MIN) S1=sqrt(x);S2=x*x; MIN=x<y?x:y
main(){
  int x,y,s2,min;
  float s1;
  printf("input two inteager numbers:\n");
  scanf("%d",&x);
  scanf("%d",&y);
  S(s1,s2,min);
  printf("s1=%.2f\ns2=%d\nmin=%d\n",s1,s2,min);
}
```

运行过程与结果为：

```
3↙
7↙
s1=1.73
s2=9
min=3
```

分析：程序第一行为宏定义，用宏名 S 表示 3 个赋值语句，3 个形参分别为 3 个赋值符左部的变量。在宏调用时，把 3 个语句展开并用实参代替形参，将计算结果送入实参之中。

语句 S(s1,s2,min)宏展开为：

```
s1=sqrt(x);s2=x*x;min=x<y?x:y;
```

7.8.3 条件编译

条件编译指令可以限定程序中的某些内容要在满足一定的条件下参与编译，以产生不同的目标代码。这对于程序的移植和调试是很有用的。

条件编译有多种格式，下面介绍常用的三种形式。

1．第一种形式

```
#ifdef 标识符
```

```
    程序段 1
[#else
    程序段 2]
#endif
```

它的功能是，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。中括号中的#else 部分可以省略。

【例 7.27】计算圆的面积或周长。

```
#include <stdio.h>
#define K
#define PI 3.1415926
#define S(r) PI*r*r
#define L(r) 2*PI*r
void main()
{ int r;
  printf ("input a number:\n");
  scanf("%d",&r);
#ifdef K
  printf("area is: %.2f\n",S(r));
#else
  printf("perimeter is: %.2f\n",L(r));
#endif
}
```

程序运行过程与结果为：

```
5↙
area is: 78.54
```

在程序第 1 行的宏定义中定义了 K，因此在条件编译时，计算并输出圆的面积。若没定义 K，则计算并输出圆的周长。

2．第二种形式

```
#ifndef 标识符
    程序段 1
[#else
    程序段 2]
#endif
```

与第一种形式的区别是将“ifdef”改为“ifndef”。它的功能是，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。这与第一种形式的功能正相反。

3．第三种形式

```
#if 常量表达式
    程序段 1
[#else
    程序段 2]
#endif
```

它的功能是，如常量表达式的值为真（非 0），则对程序段 1 进行编译，否则对程序段 2 进行编译，因此可以使程序在不同条件下完成不同的功能。

【例 7.28】求 x、y 两个数中的最大值或最小值，当 K 为 1 时，输出最大值，当 K 为 0 时，输出最小值。

```
#include <stdio.h>
#define K 1
#define MAX(x,y) (x>y?x:y)
#define MIN(x,y) (x<y?x:y)
void main()
{ int x,y;
  printf ("input two numbers:\n");
  scanf("%d",&x);
  scanf("%d",&y);
  #if K
  printf("max is: %d\n",MAX(x,y));
  #else
  printf("min is: %d\n",MIN(x,y));
  #endif
}
```

程序运行过程与结果为：

```
5↙
9↙
max is: 9
```

本例中采用了第三种形式的条件编译。在程序第 1 行的宏定义中，定义 K 为 1，因此在条件编译时，常量表达式的值为真，故计算并输出最大值。若定义 K 为 0，则输出最小值。

上面介绍的条件编译当然也可以用条件语句来实现，但是用条件语句将会对整个源程序进行编译，生成的目标代码程序很长，而采用条件编译，则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。如果条件选择的程序段很长，采用条件编译的方法是十分必要的。

7.9 综合应用

通过前面知识的学习，已经可以做稍微复杂的综合应用的题了。下面是一个具有菜单选择功能的“学生信息管理系统”的简单例子。

【例 7.29】编写简易学生成绩管理系统。要求分模块完成学生成绩的录入、查询、统计（求总分和平均分）和显示功能；用户通过选择不同的菜单项完成相应的模块功能，选择错误应提示用户重新选择；每个模块均能够正确处理输入的数据，得到正确的结果。学生成绩的存储可使用数组。

菜单如下：

学生信息管理系统

==========

1．录入学生成绩

2．统计学生成绩

3．查询学生成绩

4．显示学生成绩

0. 退出系统
请输入您的选择（0-4）：

```
#include <stdio.h>
int score[20],n=0;
void shuru()
{ int i;
  printf("      请输入录入学生成绩的人数: ");
  scanf("%d",&n);
  printf("      请输入学生成绩: \n");
  for(i=1;i<=n;i++)
  {  printf("      第 %d 号学生成绩: ",i);
     scanf("%d",&score[i]); }
}
void tongji()
{ int i;
  long s=0;
  float avg;
  printf("      已输入学生成绩的平均值为: ");
  for(i=1;i<=n;i++)
     s=s+score[i];
  avg=s/n;
  printf("%.f\n",avg);
}
void chaxun()
{ int i;
  printf("      请输入查询学生序号: ");
  scanf("%d",&i);
  printf("      第 %d 号学生成绩为: %d\n",i,score[i]);
}
void xianshi()
{ int i;
  printf("      显示已输入学生成绩: \n");
  for(i=1;i<=n;i++)
     printf("        第 %d 号学生成绩为: %d\n",i,score[i]);
}

void main()
{  int select,loop=1;
   while(loop>0)
   {
     printf("\n");
     printf("      学生信息管理系统\n");
     printf("      ================\n\n");
     printf("       1.录入学生成绩\n");
     printf("       2.统计学生成绩\n");
     printf("       3.查询学生成绩\n");
```

```
        printf("       4.显示学生成绩\n");
        printf("       0.退出系统\n\n");
        printf("       请输入您的选择（0-4）: ");
        scanf("%d",&select);
        switch (select)
         { case 1:shuru();break;
           case 2:tongji();break;
           case 3:chaxun();break;
           case 4:xianshi();break;
           case 0:printf("\n    谢谢使用本系统，再见！\n\n");loop=-1;break;
           default:printf("\n     您的选择有误，请重新选择！\n");
    }
   }
}
```

本例所用的均是学过的内容，在 5 个菜单项中，前 4 个菜单项用了函数来实现对成绩的录入、统计、查询和显示功能，主函数完成了菜单项的选择和调用功能。其中存储学生成绩的数组 score 和录入学生成绩人数 n 用了全局变量。

习　题

一、选择题

1．以下叙述中正确的是：

A．构成 C 程序的基本单位是函数

B．可以在一个函数中定义另一个函数

C．main()函数必须放在其他函数之前

D．所有被调用的函数一定要在调用之前进行定义

2．在 C 语言中，函数值类型的定义可以缺省，此时函数值的隐含类型是：

A．void　　B．int　　C．float　　D．double

3．在调用函数时，如果实参是简单变量，它与对应形参之间的数据传递方式是：

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．传递方式由用户指定

4．当调用函数时，实参是一个数组名，则向函数传送的是：

A．数组的长度　　B．数组的首地址

C．数组每一个元素的地址　　D．数组每个元素中的值

5．以下所列的各函数首部中，正确的是：

A．void play(var :Integer,var b:Integer)

B．void play(int a,b)

C．void play(int a,int b)

D．Sub play(a as integer,b as integer)

二、填空题

1．以下程序输出的最后一个值是_____。

```
#include <stdio.h>
int ff(int n)
{ static int f=1;
  f=f*n;
  return f;
}
main()
{ int i;
  for(i=1;i<=5;i++) printf("%d\n",ff(i));
}
```

2. 以下函数的功能是：求 *x* 的 *y* 次方并输出，将程序补充完整。

```
#include <stdio.h>
double fun(int x, int y)
{  int i;
   double z;
   for(i=1,z=x;i<y;i++)
   _______;
   return z;
}
main()
{   int n,x;
    double s;
    scanf("%d",&x);
    scanf("%d",&n);
    s=__________;
    printf("s=%.2f",s);
}
```

3. 设有如下宏定义，并通过宏调用实现变量 a、b 内容的交换，将程序补充完整。

```
#include <stdio.h>
#define SWAP(z,x,y) __________
main()
{   int a=5,b=16,c;
    SWAP(c,a,b);
    printf("a=%d,b=%d",a,b);
}
```

4. 若变量 n 中的值为 24，则 prnt 函数共输出_____行，最后一行有_____个数。

```
#include <stdio.h>
void prnt(int n, int aa[])
{   int i;
    for(i=0; i<n;i++)
    {
    printf("%6d", aa[i]);
    if(!(i%5)) printf("\n");
    }
    printf("\n");
}
main()
{  int aa[24]={1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4};
   prnt(24,aa);
```

```
}
```

5. 在以下程序中，主函数调用了 LineMax 函数，作用是在 N 行 M 列的二维数组中找出每一行上的最大值，将程序补充完整。

```
#include <stdio.h>
#define N 3
#define M 4
void LineMax(int x[N][M])
{   int i,j,p;
    for(i=0; i<N;i++)
    { p= x[0][0];
      for(j=0;j<M;j++)
        if(_______) p=x[i][j];
          printf("The max value in line %d is %d\n", i,p);
    }
}
main()
{   int x[N][M]={{1,5,7,4},{2,6,4,3},{8,2,3,1}};
    ____________
}
```

三、读程题

给出以下程序的输出结果。

1.

```
#include <stdio.h>
#define N 10
#define s(x) x*x
#define f(x) (x*x)
main()
{ int i1,i2;
  i1=1000/s(N); i2=1000/f(N);
  printf("%d %d\n",i1,i2);
}
```

2.

```
#include <stdio.h>
#define M(x,y,z) x*y+z
main()
{ int a=1,b=2, c=3;
  printf("%d\n", M(a+b,b+c, c+a));
}
```

3.

```
#include <stdio.h>
int func(int a,int b)
{ return(a+b);}
main()
{int x=2,y=5,z=8,r;
 r=func(func(x,y),z);
 printf("%d\n",r);
}
```

4.

```
#include <stdio.h>
void fun()
{ static int a=0;
  a+=2; printf("%d",a);
}
main()
{ int cc;
  for(cc=1;cc<4;cc++) fun();
  printf("\n");
}
```

5.

```
#include <stdio.h>
void func1(int i);
void func2(int i);
char st[]="hello,friend!";
void func1(int i)
{ printf("%c",st[i]);
  if(i<3){i+=2;func2(i);}
}
void func2(int i)
{ printf("%c",st[i]);
  if(i<3){i+=2;func1(i);}
}
main()
{ int i=0; func1(i); printf("\n");}
```

四、编程题

1. 编写一个函数，用来统计一个二维数组中非 0 元素的个数。
2. 编写一个函数，用来计算具有 10 个数的一维数组中元素的最大值、最小值与平均值。要求利用全局变量。
3. 编写一个判断素数的函数，在主函数中输入 5 个整数，若是素数，则输出该数，否则不输出。
4. 编写一个递归函数，用来解决猴子吃桃子问题。小猴在一天摘了若干个桃子，当天吃掉一半多一个；第 2 天接着吃了剩下的桃子的一半多一个；以后每天都吃剩下桃子的一半零一个，到第 7 天早上要吃时只剩下一个了，问小猴那天共摘了多少个桃子？
5. 从键盘输入 3 个数，利用宏定义求出其中的最大值。
6. 编写判断一个十进制整数是否是回文数的函数。在主函数中输入这个整数，通过被调函数来判断。