# 第 6 章　函数

函数是 C 语言程序的基本模块，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，使程序的结构清晰、减少重复编写程序的工作量、提高程序的可读性和可维护性。本章主要介绍函数的定义与调用、函数间的数据传递方法、函数的递归调用、变量的作用域和存储类别以及编译预处理命令等相关内容。

## 6.1　函数概述

在介绍 C 语言函数之前，先简单介绍模块化程序设计方法。

### 6.1.1　模块化程序设计方法

通常人们在求解一个复杂或较大规模的问题时，一般都采用逐步分解、分而治之的方法，也就是把一个大而复杂的问题分解成若干个比较容易求解的小问题，然后分别求解。人类的认知过程也遵守 Miller 法则，即一个人在任何时候都只能把注意力集中在（7±2）个知识块上。根据这一法则，程序员在设计一个大而复杂的程序时，往往也是首先把整个程序划分为若干个功能较为单一的程序模块，分别予以实现，最后把所有的程序模块像搭积木一样装配起来，完成一个完整的程序，从而达到所要求的目的。这种在程序设计中逐步分解、分而治之的策略，称为模块化程序设计方法。

如果软件可划分为可独立命名和编程的部件，则每个部件称为一个模块。模块化就是把系统划分成若干个模块，每个模块完成一个子功能，把这些模块集中起来组成一个整体，从而完成指定的功能，满足问题的要求。

例如，图书管理系统模块划分如图 6-1 所示。

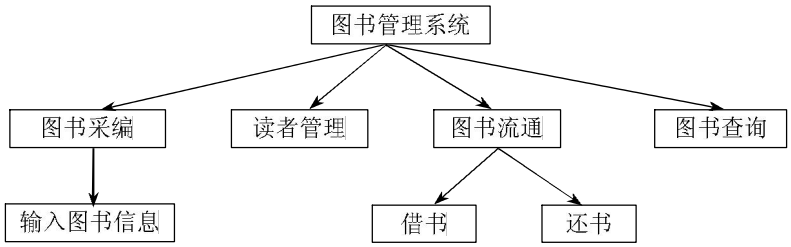



图 6-1　图书管理系统模块划分示意图

在 C 语言中，函数是程序的基本组成单位，因此可以很方便地用函数作为程序模块来实现 C 语言程序。利用函数，不仅可以实现程序的模块化，避免大量的重复工作、简化程序，提高程序的易读性和可维护性，还可以提高效率。

### 6.1.2　函数的概念

根据模块化设计的原则，一个较大的程序一般应分为若干个程序模块，每一个模块用于实现一个特定的功能。在不同的程序设计语言中，模块实现的方式有所不同。例如，在

FORTRAN 语言中，模块用子程序来实现；在 Pascal 语言中，模块用过程来实现；在 C 语言中，模块用函数来实现。

在 C 语言中，函数分为以下两种：

（1）标准库函数。这种函数用户不必定义，但可直接使用。例如 scanf()、 printf()、fabs()、sprt()、exp()、sin()、cos()等都是 C 语言中常用的库函数。

（2）用户自己定义的函数。这种函数用以解决用户的专门问题，一般由用户自己编写。

看下面的例子。

【例 6.1】 从键盘输入两个正整数 m 和 n，求 $\frac{m!}{(m-n)!}$ 的值。

其 C 程序如下：

```
#include"stdio.h"
void main()          /*主函数*/
{   int m,n;
    int p(int);      /*声明本函数中要调用的函数 p()是整型，有一个整型参数*/
    scanf("%d,%d",&m,&n);
    if(m>=n)
       printf("%d",p(m)/p(m-n));
    else
       printf("m<n!\n");
}
int p(int k)         /*计算阶乘的函数 */
{   int s,i;
    s=1;
    for(i=1;i<=k;i++)
      s*=i;
    return(s);
}
```

在这个程序中，共有两个函数：一个是主函数 main()，它的功能是从键盘输入两个正整数 m 和 n，然后当 m>=n 时，通过调用函数 p()计算并输出 p(m)/p(m-n)的值；另一个是函数 p()，它的功能是计算阶乘值，例如，p(m)是计算 m!，p(m-n)是计算(m-n)!。

（1）一个完整的 C 程序可以由若干个函数组成，其中必须有且只有一个主函数 main()。C 程序总是从主函数开始执行（不管它在程序中的什么位置），而其他函数只能被调用。

（2）一个完整 C 程序中的所有函数可以放在一个文件中；也可以放在多个文件中。例如，上述 C 程序中的两个函数可以分别放在两个文件中（主函数的文件名为 sp.c，函数 p()的文件名为 spl.c）。

```
/*主函数 main()放在文件 sp. c 中*/
#include"stdio.h"
void main()
{  int m,n;
   int p(int);
   scanf("%d,%d",&m,&n);
   if(m>=n)
      printf("%d",p(m)/p(m-n));
   else
     printf("m<n!\n"); (m-n)!
    }
    /*函数 p()放在文件 sp. c 中*/
int p(int k)
```

```
{   int s,i;
    s=1;
    for(i=1;i<=k;i++)
      s*=i;
    return(s);
}
```

如果一个 C 程序中的多个函数分别放在多个不同的文件中，在调用函数中用#include 语句将各个被调用的函数所在的文件包含进来。例如，在上面程序中的两个函数分别放在两个文件中（主函数的文件名为 sp.c，函数 p()的文件名为 spl.c），并在主函数中将被调用函数所在的文件 spl.c 包含进来，其程序如下：

```
#include"stdio.h"
#inclued"spl.c" /*在此处将文件 spl.c 包含进来 */
void main()
{   int m,n;
    int p(int);
    scanf("%d,%d",&m,&n);
    if(m>=n)
       printf("%d",p(m)/p(m-n));
    else
       printf("m<n!\n"); (m-n)!
}
/* 函数 p()放在文件 spl.c 中 */
int p(int k)
{   int s,i;
    s=1;
    for(i=1;i<=k;i++)
        s*=i;
    return(s);
}
```

C 编译系统在编译文件 sp.c 的过程中，遇到命令#inclued "spl.c"时，就将文件 spl.c 的内容包含到该命令所在的位置，并进行编译处理。

（3）C 语言中的函数没有从属关系，各函数之间互相独立，可以相互调用，但不能嵌套定义。

## 6.2 函数的定义与声明

当用户需要利用函数来完成某一个特定的任务，又没有相应的库函数可以使用时，就必须自定义一个函数来完成任务。要在 C 语言中使用用户自定义的函数，必须遵循先定义、后声明、再使用的步骤，即首先应定义好函数的数据类型、存储类型和函数体，然后才能使用。

下面是一个函数声明与定义之间及调用关系的例子。

【例 6.2】 输入三个整数，求三个整数中的最大值。

```
#include"stdio.h"
void main()                          /* 主函数 */
{
    int n1,n2,n3,nmax;
    int max3(int x,int y,int z);    /* 函数的声明 */
```

```
    printf("请输入 n1,n2,n3 的值:\n");
    scanf("%d,%d,%d",&n1,&n2,&n3);
    nmax=max3(n1,n2,n3);
    printf("max=%d\n ",nmax);
}
int max3(int x,int y,int z)          /* 函数的定义 */
{
    int m;
    if(x>y)
      m=x;
    else
      m=y;
    if(z>m)  m=z;
    return m;
}
```

### 6.2.1　函数定义

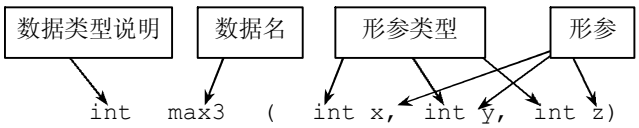
函数定义的一般形式：

[函数类型] 函数名 ([形式参数表])　　/* 函数头 */

{　[声明部分]

　[执行语句]

}




```
int   max3   (   int x,  int y,  int z)
 {
       int m;
       if(x>y)
          m=x;
       else
          m=y;
       if(z>m)  m=z;
       return m;
  }
```

说明：一个函数（定义）由函数头（函数首部）和函数体两部分组成。

（1）函数头（首部）：说明函数类型、函数名称及参数。

1）函数类型：函数返回值的数据类型，可以是基本数据类型也可以是构造类型。如果省略则默认为 int 类型，如果不返回值，则定义为 void 类型。

2）函数名：给函数取的名字，以后用这个名字调用。函数名由用户命名，命名规则与标识符相同。

3）函数名后面是形式参数表，也可以没有参数，但“( )”号不能省略，这是格式的规定。形式参数表说明形式参数的类型和形式参数的名称，各个形式参数之间用“,”分隔。

（2）函数体：函数头下方用一对{}括起来的部分。如果函数体内有多个{}，最外层是函数体的范围。函数体一般包括声明部分和执行部分。

1）声明部分：定义本函数所使用的变量和进行有关声明（如函数声明）。

2）执行部分：程序段，即由若干条语句组成的命令序列（可以在其中调用其他函数）。

**注意**：函数不能单独运行，函数可以被主函数或其他函数调用，也可以调用其他函数，但是不能调用主函数。

在 C 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 main 之前，也可放在 main 之后。

现在以例 6.2 为例从函数定义、函数声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 11 行至最后为 max3 函数定义。进入主函数后，因为准备调用 max3 函数，故先对 max3 函数进行声明（程序第 5 行）。函数定义和函数声明并不是一回事，在后面还要专门讨论。可以看出函数声明与函数定义中的函数头部分相同，但是末尾要加分号。程序第 8 行为调用 max3 函数，并把 n1，n2，n3 中的值传送给 max3 的形参 x，y，z。max3 函数执行的结果 m 将返回给变量 nmax，最后由主函数输出 nmax 的值。

### 6.2.2 函数的参数和返回值

前面已经介绍过，函数的参数分为形式参数和实际参数两种。

形式参数（形参）：函数定义时设定的参数。如例 6.2 中，函数头 int max3(int x,int y,int z) 中 x，y，z 就是形参，都是整型。

实际参数（实参）：调用函数时所使用的实际的参数。如例 6.2 中，主函数中调用 max3 函数的语句 nmax=max3(n1,n2,n3)中，n1，n2，n3 就是实参，都是整型。

形参和实参的功能是进行数据传递。发生函数调用时，主调函数把实参的值传递给被调函数的形参，从而实现主调函数向被调函数的数据传递。

C 语言可以从函数（被调用函数）返回值给调用函数（这与数学函数相当类似）。在函数内是通过 return 语句返回值的。使用 return 语句能够返回一个值或不返回值（此时函数类型是 void）。return 语句的格式为：

return [表达式];或 return(表达式);

说明：

（1）函数的类型就是返回值的类型，return 语句中表达式的类型应该与函数类型一致。如果不一致，以函数类型为准（赋值转化）。

（2）函数类型省略，默认为 int。

（3）如果函数没有返回值，函数类型应当说明为 void（空类型）。

### 6.2.3 函数的声明

函数定义的位置可以在调用它的函数之前，也可以在调用它的函数之后，甚至可位于其他的源程序模块中。

若函数定义位置在前，函数调用在后，不必声明，编译程序产生正确的调用格式。

若函数定义在调用它的函数之后或者函数在其他源程序模块中，且函数类型不是整型，这时，为了使编译程序产生正确的调用格式，可以在函数使用前对函数进行声明。这样不管函数在什么位置，编译程序都能产生正确的调用格式。

函数声明的格式为：

函数类型 函数名([形式参数表]);

C 语言的库函数就是位于其他模块的函数，为了正确调用，C 编译系统提供了相应的.h 文件。.h 文件内许多都是函数声明，当源程序要使用库函数时，就应当包含相应的头文件。

## 6.3　函数的调用

一个函数调用另一个函数称为函数调用，其调用者称为主调函数，被调用者称为被调函数。在 C 语言中可以根据需要调用任何函数来实现某种功能。

### 6.3.1　调用函数的一般形式

C 语言中，函数调用的一般形式为：

函数名([实参表列]);

说明：

（1）无参函数调用没有参数，但是“()”不能省略，有参函数若包含多个参数，各参数用“,”分隔，实参参数个数与形参参数个数相同，类型一致或赋值兼容。

（2）函数调用可以出现的位置如下：

1）以单独语句形式调用（注意后面要加一个分号，构成语句）。以语句形式调用的函数可以有返回值，也可以没有返回值。

例如：

```
printf("max=%d",nmax);
swap(x,y);
puts(s);
```

2）在表达式中调用（后面没有分号）。在表达式中的函数调用必须有返回值。

例如：

```
if (strcmp(s1,s2)>0)……  /*函数调用 strcmp()在关系表达式中*/
nmax=max(n1,n2,n3);      /*函数调用 max()在赋值表达式中，“;”是赋值表达式作为语*/
                         /*句时加的，不是 max 函数调用的*/
fun1(fun2());            /*函数调用 fun2()在函数调用表达式 fun1()中。函数调用 fun2()*/
                         /*的返回值作为 fun1 的参数*/
```

调用函数时需要注意下列事项：

- 被调用函数必须是已经存在的函数（库函数和用户自定义函数）。
- 如调用库函数，在文件的开头必须使用#include 命令将库函数相应的头文件包含进来。
- 如调用用户自定义函数，在调用之前必须对其进行声明。

### 6.3.2　调用函数时数据的传递

函数是相对独立的，但并不是孤立的，它们通过调用时参数传递、函数的返回值和全局变量（后面介绍）来相互联系。

前面已经介绍过，函数的参数分为形参和实参两种。形参是被调函数中的参数，实参是主调函数中的参数。形参和实参的功能是进行数据传递。在调用函数时，主调函数和被调函数之间有数据的传递——实参传递给形参。具体的传递方式有两种：

（1）值传递方式（传值）：将实参单向传递给形参的一种方式。

（2）地址传递方式（传址）：将实参地址单向传递给形参的一种方式。

目前，我们所使用的形参都是变量参数。在 C 程序中，采用变量作形参时，实参和形参是按传值方式相结合的，也称为传值调用方式。

在传值调用时，函数的形参和实参具有以下特点：

（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，形参变量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，以便把这些值传递给形参。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，否则会发生“类型不匹配”的错误。

（4）函数调用中发生的数据传递是单向的。即只能把实参的值传送给形参，而不能把形参的值传递给实参，如图 6-2 所示。

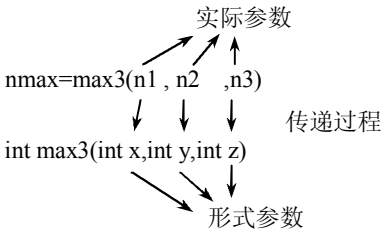



图 6-2　传值调用

【例 6.3】 定义函数，求 $\sum_{i=1}^{n} i$ 的值。

```
#include"stdio.h"
void main()
{
    int n;
    void s(int n);
    printf("请输入 n 的值:\n");
    scanf("%d",&n);
    s(n);
    printf("n=%d\n",n);
}
void s(int n)
{
    int i;
    for(i=n-1;i>=1;i--)
       n=n+i;
    printf("n=%d\n",n);
}
```

运行结果如下：

```
n=100
n=5050
n=100
```

本程序中定义了一个函数 s，该函数的功能是求 $\sum_{i=1}^{n} i$ 的值。在主函数中输入 n 值，并作为实参，在调用时传递给 s 函数的形参量 n。

注意，本例的形参变量和实参变量的标识符都为 n，但这是两个不同的量，各自的作用域不同。在主函数中用 printf 语句输出一次 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。从运行情况看，输入 n 值为 100，即实参 n 的值为 100。把此值传给函数 s 时，形参 n 的初值也为 100，在执行函数过程中，形参 n 的值变为 5050。返回主函数之后，输出实参 n 的值仍为 100。可见实参的值不随形参的变化而变化。

### 6.3.3　函数的嵌套调用

C 语言中函数定义都是互相平行、独立的，也就是说在定义函数时，一个函数内不能包含另一个函数。

C 语言不能嵌套定义函数，但可以嵌套调用函数，也就是说，在调用一个函数的过程中，又调用另一个函数。其关系如图 6-3 所示。

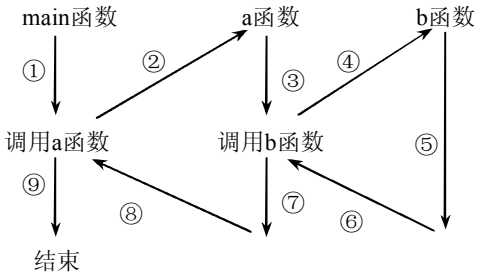



图 6-3　嵌套调用

图 6-3 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 6.4】 计算 $s=2^2!+3^2!$。

分析：本题可编写两个函数，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。

程序如下：

```
#include"stdio.h"
long f1(int p)
{
    int k;
    long r;
    long f2(int q);
    k=p*p;
    r=f2(k);
    return r;
}
long f2(int q)
{
    long c=1;
    int i;
    for(i=1;i<=q;i++)
       c=c*i;
```

```
    return c;
}
void main()
{
    int i;
    long s=0;
    for(i=2;i<=3;i++)
       s=s+f1(i);
    printf("s=%ld\n",s);
}
```

在程序中，函数 f1 和 f2 均为长整型，都在主函数之前定义，故不必再在主函数中对 f1 和 f2 加以说明。在主程序中，执行循环程序依次把 i 值作为实参调用函数 f1 求 $i^2$ 值。在 f1 中又发生对函数 f2 的调用，这时把 $i^2$ 的值作为实参去调用 f2，在 f2 中完成求 $i^2$!的计算。f2 执行完毕把 c 值（即 $i^2$!）返回给 f1，再由 f1 返回主函数实现累加。至此，由函数的嵌套调用实现了题目的要求。由于数值很大，所以函数和一些变量的类型都说明为长整型，否则会造成溢出。

### 6.3.4 函数的递归调用

一个函数在它的函数体内调用它自身的过程称为递归调用，递归调用表现为直接调用自己或间接调用自己两种方式，也就是一个函数的执行过程中出现直接或间接调用该函数自身的行为。前者称为直接递归调用，后者称为间接递归调用。这种函数称为递归函数。

**【例 6.5】** 有 5 个人坐在一起，问第 5 个人多少岁？他说比第 4 个人大 2 岁。问第 4 个人岁数，他说比第 3 个人大 2 岁。问第 3 个人，又说比第 2 个人大 2 岁。问第 2 个人，说比第一个人大 2 岁。最后问第 1 个人，他说是 10 岁。请问第 5 个人多大。

这是一个递归问题。想求第 5 个人的年龄，就必须先知道第 4 个人的年龄，而第 4 个人的年龄也不知道，要想求第 4 个人的年龄必须先知道第 3 个人的年龄，而第 3 个人的年龄又取决于第 2 个人的年龄，第 2 个人的年龄取决于第 1 个人的年龄。而且每一个人的年龄都比其前 1 个人的年龄大 2。如果 age 是年龄函数，age(n)代表第 n 个人的年龄，可以用下面的式子表示上述关系。

```
age(5)=age(4)+2
age(4)=age(3)+2
age(3)=age(2)+2
age(2)=age(l)+2
age(l)= 10
```

从上可以看出，求第 n（n>1）个人年龄的方法是相同的。因此，可以用一个函数表示上述关系。图 6-4 表示求第 5 个人年龄的过程。

从图 6-4 可知，求解可分成两个阶段：第一阶段是“回溯”，即将第 n 个人的年龄表示为第(n-1)个人年龄的函数，而第(n-l)个人的年龄仍然不知道，还要“回推”到第(n-2)个人的年龄……直到第 1 个人的年龄。此时 age(1)已知，不必再向前推了。然后开始第二阶段，采用递推方法，从第 1 个人的已知年龄推出第 2 个人的年龄（12 岁），从第 2 个人的年龄推出第 3 个人的年龄（14 岁）……一直推算出第 5 个人的年龄（18 岁）为止。也就是说，一个递归问题可以分为“回溯”和“递推”两个阶段。要经历若干步才能求出最后的值。显而易见，如果要求递归过程不是无限制地进行下去，必须有一个递归过程的边界条件。例如，age(1)=10，就是递归的边界条件。

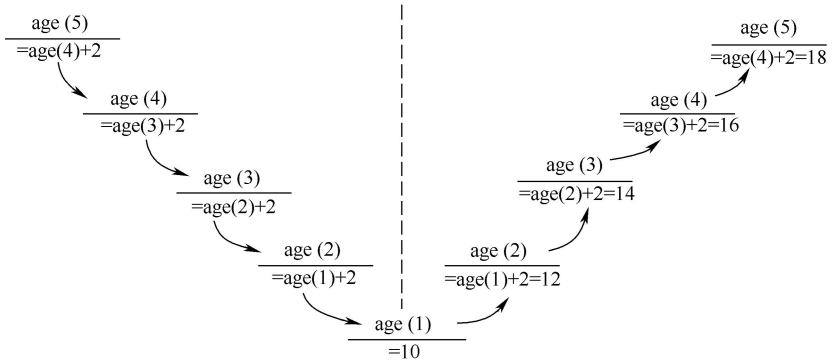



图 6-4　递归调用

可以用一个函数来描述上述递归过程：

```
int age(int n)        /*求年龄的递归函数*/
{  int c;             /*C 用作存放函数的返回值的变量*/
   if(n==1)
      c=10;
   else
      c=age(n-1)+2;
   return(c);
}
```

用一个主函数调用 age 函数，求得第 5 个人的年龄：

```
#include"stdio.h"
void main()
{
   printf("%d\n",age(5));
}
```

主函数的位置可以在 age 函数之后（如本例上面的那样），这时在 main 函数中不必对 age 函数进行声明。

运行结果如下：

```
18
```

main 函数只有一个语句。整个问题的求解全靠一个 age(5)函数调用来解决。函数的调用过程如图 6-5 所示。

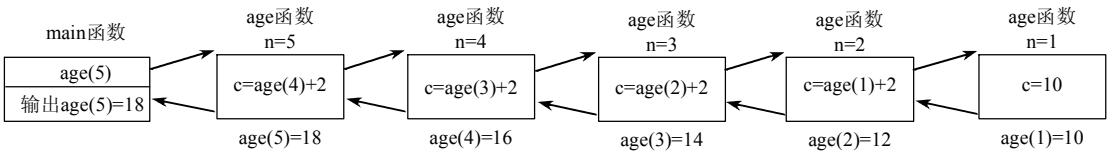



图 6-5　调用过程

从图 6-5 可以看出：age 函数共被调用 5 次，即 age(5)，age(4)，age(3)，age(2)，age(1)。其中 age(5)是 main 函数调用的，其余 4 次是在 age 函数中调用的，即递归调用 4 次。请读者仔细分析调用的过程。应当强调说明的是某一次调用 age 函数时并不是立即得到 age(n)的值，而是一次又一次地进行递归调用，到 age(1)时才有确定的值，然后再递推出 age(2)，age(3)，age(4)，age(5)。请读者将程序和图 6-3 和图 6-4 结合起来认真分析。

【例 6.6】 用递归法求 n!。

分析：用递归法计算 n!可用下述公式表示：

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

当 n 大于等于 2 的时候，欲求 n 的阶乘，只要先求出 n-l 的阶乘，把所得结果乘以 n 即为 n 的阶乘。至于 n-l 的阶乘是多少，在这一步可先不考虑，设它为 p。则

$$n!=n\times p$$

这是一步就可求出的问题，但这时 p 还不是一个确定的数，还要等待返回结果。

接下来求 p=(n-l)!，再把它进行分解。它等于 n-l 去乘 n-2 的阶乘，即 p= (n-l)×(n-2)！，设(n-2)！=q，则 p=(n-l)×q。接下来的问题是再去求 q。按照上述方法继续类似的操作，问题一步步地化简，最终一定能达到求 1 的阶乘的地步，这时就称达到了边界条件，可以直接求解了。任何递归调用都有一个边界条件，否则递归调用将会无限地进行下去。递归调用关系如图 6-6 所示。

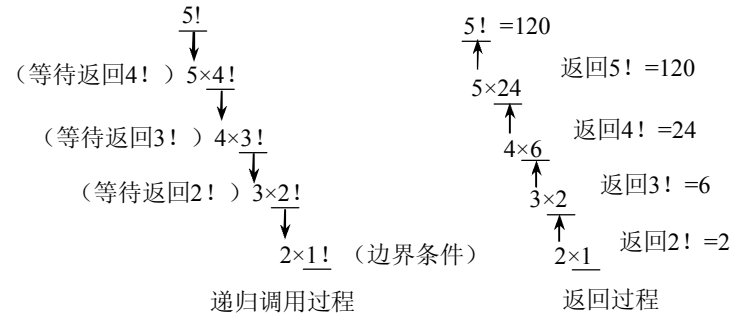



图 6-6　递归调用

程序如下：

```
#include"stdio.h"
long ff(int n)
{
    long f;
    if(n==0||n==1)
        f=1;
    else
        f=ff(n-1)*n;
    return(f);
}
void main()
{
    int n;
    long y;
    printf("请输入 n 的值:\n");
    scanf("%d",&n);
    if(n<0)
        printf("n<0,input error!");
    else
       {
         y=ff(n);
         printf("%d!=%ld",n,y);
        }
}
```

程序中给出的函数 ff 是一个递归函数。当 n>=0 时，主函数调用 ff 后即进入函数 ff 执行，如果 n<0，n=0 或 n=1 时都将结束函数的执行，否则就递归调用 ff 函数自身。由于每次递归调

用的实参为 n-1，即把 n-1 的值赋予形参 n，最后当 n-1 的值为 1 时再作递归调用，形参 n 的值也为 1，将使递归终止。

从前面的分析可以看出，递归调用过程包含两个阶段：

（1）递推阶段：将原问题不断地分解为新的子问题，逐渐从未知的向已知的方向推测，最终达到已知的条件，即递归结束条件，这时递推阶段结束。

（2）回归阶段：从已知条件出发，按照“递推”的逆过程，逐一求值回归，最终到达“递推”的开始处，结束回归阶段，完成递归调用。

递归过程不应无限制地进行下去，在设计递归函数时应当考虑到递归的终止条件，在本例中，下面就是使递归终止的条件：

```
if(n==0||n==1)
    f=1;
```

所以，任何有意义的递归总是由递归方式与递归终止条件两部分组成的。

递归是一种非常有用的程序设计技术。当一个问题蕴含递归关系且结构比较复杂时，采用递归算法往往比较自然、简洁、容易理解。请读者仔细阅读以上程序，理解递归算法的思路，学会用递归方法解决问题。

## 6.4　局部变量和全局变量

在讨论函数的形参变量时曾经提到，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为变量的作用域。不仅对于形参变量，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：局部变量和全局变量。

### 6.4.1　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：

```
#include"stdio.h"
int f1(int a)              /* 函数 f1 */
   {
      int b,c;             /* 变量 a，b，c 作用域 */
      ……
}
int f2(int x)              /* 函数 f2 */
{
      int y,z;             /* 变量 x，y，z 作用域 */
      ……
}
void main()                /* 函数 main */
{
      int m,n;             /* 变量 m，n 作用域 */
      ……
}
```

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，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：

```
void main()
{
       int s,a;
       ……
      {
         int b;
         s=a+b;
          ……
       }
      ……
}
```

b 的作用域

s，a 的作用域

再来看下面一个例子，分析变量的作用域及输出结果。

【例 6.7】 写出程序的执行结果。

```
void main()
{
    int i=2,j=3,k;
    k=i+j;
   {
      int k=8;
      printf("%d\n",k);
    }
    printf("%d\n",k);
}
```

运行结果如下：

```
8
5
```

本程序在 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。应该注意这两个 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。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。

### 6.4.2 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，而属于

一个源程序文件。其作用域是从定义变量的位置开始到本源文件结束。

例如：

```
#include"stdio.h"
int a,b;          /*外部变量*/
void f1()         /*函数 f1*/
{
   ……
}
float x,y;        /*外部变量*/
int f2()          /*函数 f2*/
{
   ……
}
void main()       /*主函数*/
{
   ……
}
```

全局变量 x，y 的作用域

全局变量 a，b 的作用域

从上例可以看出 a、b、x、y 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，故在 f2 与 main 内可以使用。a，b 定义在源程序最前面，因此在 f1、f2 及 main 内不加说明也可以使用。

如果全局变量在文件的开头定义，则在整个文件范围内均可以使用该全局变量；如果不在文件的开头定义，又想在定义点之前使用该全局变量，需要用 extern 进行声明。

变量的声明和定义是有区别的：变量的定义即为变量分配存储单元；变量的声明即说明变量的性质，并不分配存储空间。

全局变量是实现函数之间数据通信的有效手段。全局变量可加强函数模块之间的数据联系，但是又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。

【例 6.8】 输入正方体的长宽高 l，w，h，求体积及三个面的面积。

```
int s1,s2,s3;
int vs(int a,int b,int c)
{
    int v;
    v=a*b*c;
    s1=a*b;
    s2=b*c;
    s3=a*c;
    return v;
}
void main()
{
   int v,l,w,h;
   printf("请输入长,宽,高的值:\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);
}
```

分析：

（1）程序中定义了 3 个全局变量 s1，s2，s3，用来存放 3 个面积，其作用域为整个程序。

（2）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。

（3）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，用全局变量是一种很好的方式。本例中，如不使用全局变量，在主函数中就不可能取得 v，s1，s2，s3 四个值。而采用全局变量，在函数 vs 中求得的 v，s1，s2，s3 值在 main 函数中仍然有效。因此全局变量是实现函数之间数据通信的有效手段。

对于全局变量还有以下几点说明：

（1）对于局部变量的定义和声明，可以不加区分。而对于全局变量则不然，全局变量的定义和全局变量的声明并不是一回事。全局变量定义必须在所有的函数之外，且只能定义一次。其一般形式为：

类型说明符　变量名,变量名……

全局变量声明的一般形式为：

extern　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，全局变量在定义可作初始赋值，但不能在声明时赋初始值，全局变量的声明只是表明在函数内要使用该全局变量。

【例 6.9】 编写程序，输入两个数，调用函数找出最大值。

```
#include"stdio.h"
int mymax(int x,int y)
{
    int z;
    if(x>y)
        z=x;
    else
        z=y;
    return z;
}
void main()
{
    extern int a,b;    /* 全局变量声明 */
    printf("请输入 a,b 的值:\n");
    scanf("%d,%d",&a,&b);
    printf("max=%d\n",mymax(a,b));
 }
int a,b;             /* 全局变量定义 */
```

全局变量只在所有函数之外定义一次。全局变量可以声明多次，哪个函数内要用到在其后面定义的变量，就需要在该函数内对该全局变量进行声明。

（2）在同一个源文件中，允许全局变量与局部变量同名。在局部变量的作用范围内，全局变量被“屏蔽”，即它不起作用。

【例 6.10】 全局变量与局部变量同名。

```
#include"stdio.h"
int a=3,b=5;             /* a，b 为全局变量 */
int mymax(int a,int b)  /* a，b，c 为局部变量 */
{
   int c;
   c=a>b?a:b;
   return(c);
}
```

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

运行结果如下：

```
8
```

（3）如果没有全局变量，函数只能通过参数与外界发生数据联系，有了全局变量以后，增加了一条与外界传递数据的渠道。这种方法有利有弊。全局变量作为公共信息的一种载体，虽然给程序设计带来一些方便，但也会产生一些副作用，见下面的例子。

【例 6.11】 全局变量的副作用。

```
#include"stdio.h"
int i;
void main()
{
    void prt();
    for(i=0;i<5;i++)
    prt();
}
void prt()
{
    for(i=0;i<5;i++)
       printf("%c",'*');
    printf("\n");
}
```

运行结果如下：

```
*****
```

而程序设计者本来试图输出一个由“*”组成的 5×5 的方阵：

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

上述程序却只输出了一行。原因是 prt()执行一次后，i 已变为 5，返回 main()后，便退出 for 结构。

这是一个极小的例子。随着程序规模增大，使用的全局变量增多，全局变量所引起的副作用会令人防不胜防，难以控制。各模块之间除了用参数传递信息之外，还增加了许多意料之外的渠道，造成模块之间的联系太多，对外部的依赖太多，降低了模块的独立性，给设计、调试、排错和维护都带来困难。此外，它无论是否使用，程序执行时都占用固定的空间。因此，在程序设计时应有限制地使用全局变量。

## 6.5　变量的存储属性

### 1. 用户程序的存储分配

一般来说，用户程序在计算机中的存储分配图如 6-7 所示。其中存储单元，如函数形参、自动变量、函数调用时的现场保护和地址返回等。

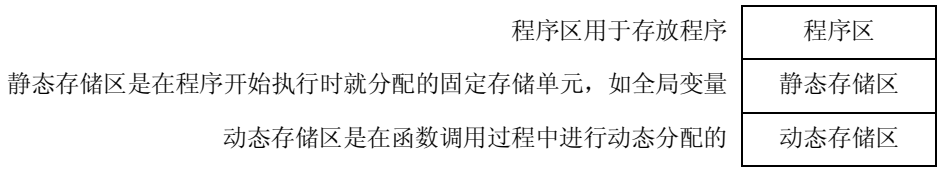



图 6-7　用户程序在计算机中的存储分配

2. 变量的存储类型

在 C 语言中，每一个变量和函数都有两个属性：操作属性和存储属性。数据类型是变量的操作属性，变量的存储属性包括变量的存储器类型、变量的生存期和变量的作用域。

前面已经介绍了，从变量的作用域（即从空间）角度来分，可以分为全局变量和局部变量。

从另一个角度，从变量值存在的作用时间（即生存期）角度来分，可以分为永久存储和动态存储。采用永久存储的变量在编译时分配存储单元，程序执行开始后这种变量即被创建，程序执行结束才被撤销。这种变量的生存期为程序执行的整个过程，在该过程中占有固定的存储空间。而采用动态存储的变量只在程序执行的某一段时间内存在。例如，函数的形参和在函数体或分程序中定义的变量，只是在程序进入该函数或分程序时才分配存储空间，当该函数或分程序执行完后存储空间又被撤销了。

众所周知，计算机的存储器分为内存储器（主存）和外存储器（辅存）。实际上，除主（内）存与辅（外）存外，CPU 中还有一个小小的临时存储器称为寄存器，用以存储一些反复被加工的数据。寄存器的存取速度比主存快。C 语言允许程序员区分是在主存中还是在寄存器中开辟变量存储空间。

在 C 语言中，用“存储属性”来表示以上三个方面的属性，并且把它们分为 4 类：register，auto，static，extern。

存储属性是变量的重要属性。C 语言要求，在定义一个变量时，除了指定其数据类型以外，还可以指定其存储属性。例如：

```
register int a;
```

表示 a 是一个整型变量，它存储在寄存器内而不存储在主存中。以前只是简单地写“int a;”，这隐含表示存储在主存中。所以除了 register 类型，其余三种类型均存储在主存中。

下面分别介绍这些存储属性。

### 6.5.1　自动变量（auto）

自动变量为局部变量，用说明符 auto 进行说明，是 C 语言程序中使用最广泛的一种变量。当程序的一个局部要使用某些自动变量时，应当在使用之前先按如下形式进行说明：

[auto]  数据类型  变量名[= 初值表达式],…;

其中，方括号表示可省略。如果省略 auto，系统隐含认为此变量为 auto。过去使用的变量，实际上都是 auto 类型的变量。

下面对自动变量作进一步说明。

（1）自动变量是局部变量。自动变量的作用域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，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。

【例 6.12】 有下列函数：

```
#include"stdio.h"
void main()
{/*          ①          */
   int x=1;
   {/*       ③          */
      void prt();
      int x=3;
      prt();
      printf("2nd x=%d\n",x);
    }/*      ④          */
    printf("1st x=%d\n",x);
}/*          ②          */
void prt()
{/*          ⑤          */
    int x=5;
    printf("3th x=%d\n",x);
}/*          ⑥          */
```

程序中先后定义了三个变量 x，它们都是自动变量，都只在本函数或复合语句中有效。

在 main 函数中定义的变量 x 在①到②范围内有效，可以被引用。但是当程序执行到③时，又定义了一个 x 变量，它的作用域为该复合语句内，即③到④。现在发生了一个矛盾：外层的 x 和内层的 x 同名，究竟它们是否为同一个变量？如果不是同一个变量，那么每一个的作用范围是什么？在③到④范围内起作用的是哪一个 x？C 语言规定，如果内层与外层有相同名字的变量，则在内层范围内只有内层的变量有效，外层的同名变量在此范围内无效，或者说，外层的变量被内层的同名变量“屏蔽”掉了。因此在③到④复合语句内输出的 x 的值是 3，而 main 函数中最后一个语句输出 x 的值为 1。因为外层的 x 与内层的 x 不是同一个变量，内层的 x 是 3，而外层的 x 仍为 1。prt 函数中定义的变量 x 只在该函数中有效，它与前面两个 x 互不相干。

运行结果如下：

```
3th x=5
2nd x=3
1st x=1
```

（4）在对自动变量赋值之前，它的值是不确定的。

【例 6.13】 使用未赋初值的自动变量。

```
#include"stdio.h"
void main()
{
   int i;
   printf("i=%d\n",i);
}
```

运行结果如下：

```
i=62
```

这里 62 是一个不可预知的数，由 i 所在的存储单元中当时的状态决定。

因此，对于自动变量，必须对其赋初值后，才能引用它。

最后应当指出，函数的形参也是一种自动变量，但是在说明时不加 auto。

### 6.5.2 寄存器变量（register）

寄存器变量具有与自动变量完全相同的性质。为了提高效率，C 语言允许将局部变量的值放在 CPU 的寄存器中，这种变量叫“寄存器变量”，用关键字 register 声明。

【例 6.14】 使用寄存器变量。

```
#include"stdio.h"
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);
}
void main()
{
   int i;
   for(i=0;i<=5;i++)
      printf("%d!=%d\n",i,fac(i));
}
```

由于频繁使用变量 i，故将它放在寄存器中。

说明：

（1）只有局部自动变量和形式参数可以作为寄存器变量。

（2）一个计算机系统中的寄存器数目有限，不能定义任意多个寄存器变量。

### 6.5.3 静态变量（static）

有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，这时就应该指定局部变量为“静态局部变量”，用关键字 static 进行声明。

静态变量的定义采用下面的格式：

static　数据类型　变量名[= 初始化常量表达式],…;

下面对静态变量作进一步说明：

（1）静态变量的存储空间在程序的整个运行期间是固定的。一个变量被指定为静态的，在编译时就为其分配存储空间，程序一开始执行便被建立，直到该程序执行结束都是存在的。

（2）静态变量的初始化是在编译时进行的。在定义时只能使用常量或常量表达式进行显式初始化。未显式初始化时，编译时将把它们初始化为 0（对 int 型）或 0.0（对 float 型）。

严格地讲，初始化指的是在程序运行前由编译器给变量的初始值。由于自动变量是在程序运行过程中被创建的，因而没有初始化的问题，只有静态变量和外部变量是在编译时被创建的，才有初始化问题。所以把对自动变量称为“赋初值”，而把对静态变量和外部变量称为“初始化”，以示区别。然而，在不太严格的情况下，人们也统统把用声明语句给定初值称为“初始化”。

（3）在函数多次被调用的过程中，静态局部变量的值具有可继承性。

【例 6.15】 考察静态局部变量的值。

```
#include"stdio.h"
int f(int a)
{
   auto int b=0;
```

```
    static int c=3;    /* 静态变量 */
    b=b+1;
    c=c+1;
    return(a+b+c);
}
void main()
{
   int a=2,i;
   for(i=0;i<3;i++)
      printf("%d\n",f(a));
}
```

运行结果如下：

```
7
8
9
```

除了静态局部变量之外，还有静态外部变量，这将在下面介绍。

### 6.5.4　外部变量

外部变量（即全局变量）是在函数的外部定义的，其作用域为从变量定义处开始，到本程序文件末尾。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的末尾。如果在定义点之前的函数想引用该外部变量，则应该在引用之前用关键字 extern 对该变量作“外部变量声明”，表示该变量是一个已经定义的外部变量。有了此声明，就可以从“声明”处起，合法地使用该外部变量。

【例 6.16】 用 extern 声明外部变量，扩展程序文件中的作用域。

```
#include"stdio.h"
int mymax(int x,int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}
void main()
{
    extern a,b;
    printf("%d\n",mymax(a,b));
}
int a=13,b=-8;
```

在本程序文件的最后一行定义了外部变量 a，b，但由于外部变量定义的位置在 main 函数之后，因此在 main 函数中不能引用外部变量 a，b。现在在 main 函数中用 extern 对 a 和 b 进行“外部变量声明”，就可以从“声明”处起，合法地使用该外部变量 a 和 b。

说明：

（1）还可以将外部变量的作用域扩充到其他文件。这时在需要用到这些外部变量的文件中，对变量用 extern 作声明即可。例如：

```
file1.c                          file2.c
int x,y;                         extern int x,y;
char ch;                         extern char ch;
void main()                      f1()
{                                {
```

```
    ……                                  printf("%d,%d\n",x,y);
    x=12;                               ……
    y=24;                               ch='a';
    f1();                               ……
    printf("%c",ch);                }
}
```

在 file2.c 文件中没有定义变量 x，y，ch，而是用 extern 声明 x，y，ch 是外部变量，因此在 file1.c 中已定义的变量在 file2.c 中可以引用。x，y 在 file1.c 中被赋值，它们在 file2.c 中也作为全局变量，因此 printf 语句输出 12 和 24。同样，在 file2.c 中对 ch 赋值'a'，在 file1.c 中也能引用它的值。当然，要注意操作的先后顺序，只有先赋值才能引用。

（2）限定本文件的外部变量只在本文件中使用。如果有的外部变量只允许本文件使用而不允许其他文件使用，则可以在此外部变量前加一个 static，使其有限局部化，称作静态外部变量。例如：

```
static int a=3,b=5;
void main()
{
  ……
}
void f1()
{
  ……
}
void f()
{
  ……
}
```

在本文件中，a，b 为全局变量，但作用域也仅限于本文件。

使用静态外部变量的好处是：当多人分别编写一个程序的不同文件时，可以按照需要命名变量，而不必考虑是否会与其他文件中的变量同名，以保证文件的独立性。

由此，可以将这四类存储属性的性质总结如表 6-1 所示。

**表 6-1　四类存储属性的性质**

| 存储属性 | register | auto | static | extern |
|---|---|---|---|---|
| 存储位置 | 寄存器 | 主存 | | |
| 生存期 | 动态存储 | | 永久存储 | |
| 作用域 | 局部 | | 局部或全局 | 全局 |

## 6.6　编译预处理

编译预处理是在编译前对源程序进行的一些预处理。预处理由编译系统中的预处理程序按源程序中的预处理命令进行。

C 语言的预处理命令均以“#”打头，末尾不加分号，以区别于 C 语句。它们可以出现在程序中的任何位置，其作用域是自出现点到所在源程序的末尾。前面我们已经使用过两个预处理命令：#define 和#include。

编译预处理是 C 语言的一个重要特点，合理地使用预处理功能编写的程序便于阅读、修

改、移植和调试，也有利于模块化程序设计。本节介绍常用的几种预处理功能。

### 6.6.1　宏定义

在 C 语言源程序中允许用一个标识符来表示一个字符串，称为“宏”。被定义为“宏”的标识符称为“宏名”。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”。

宏定义是由源程序中的宏定义命令完成的，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　字符串

其中，作为宏名的标识符习惯上用有意义且易理解的大写字母来表示，“字符串”可以是常数、表达式或格式串等。宏定义一般写在文件开头函数体的外面，有效范围是从定义宏命令之后到遇到终止宏定义命令#undef 为止，否则其作用域将一直到源文件结束。

例如：

```
#define PI 3.1415926
```

即定义了宏名 PI 来代表 3.1415926。在编译预处理时，系统将把该命令之后作用域之内的所有 PI 都自动用 3.1415926 代换，即进行宏展开。实际上这就是前面介绍过的符号常量的定义形式。

使用宏定义，一方面可以减少频繁使用的字符串的重复书写；另一方面也使得修改重复使用的字符串的工作变得简单了，因为只需在宏定义处修改一次即可。

对于宏定义还要说明以下几点：

（1）宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名，这只是一种简单的代换，字符串中可以含任何字符，可以是常数，也可以是表达式，预处理程序对它不作任何检查。如有错误，只能在编译已被宏展开后的源程序时发现。

例如：

```
#define PI  "3.1415926"
```

那么，在预处理时将把 PI 替换为带双引号的字符串"3.1415926"，代入表达式中就会出错。

（2）如果在一行中写不下整个宏定义，需要用两行或更多行来书写时，只需在每行的最后一个字符的后面加上反斜杆“\”，并在下一行的最开始接着写即可。

（3）宏定义必须写在函数之外，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。

例如：

```
#define PI 3.14159
void main()
{
  ……
}
#undef PI
f1()
{
  ……
}
```

表示 PI 只在 main 函数中有效，在 f1 中无效。

（4）宏名在源程序中若用引号括起来，则预处理程序不对其作宏代换。

例如，如果程序中有以下语句：

```
printf("PI=",PI);
```

在预处理时，将只对第二个 PI 进行代换，而对第一个双引号中的 PI，系统并不对其作代换。执行该语句会输出：

```
PI=3.1415926
```

（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。

例如：

```
#define R 5.6
#define PI 3.1415926
#define  L  2*PI*R
#define S PI*R*R          /* PI，R 是已定义的宏名*/
```

在宏展开时，编译器会把程序中的 R 用 5.6 来代换，把 PI 用 3.1415926 来代换，而将 S 用 3.1415926*5.6*5.6 来代换。

【例 6.17】 已知半径为 5.0，计算以此为半径的圆的周长和面积，以及圆球体的体积。

```
#include"stdio.h"
#define  R  5.0
#define  PI  3.1415926

#define  S  PI*R*R
#define  V  4.0/3.0*PI*R*R*R
void main()
{
  printf("L=%f\nS=%f\nV=%f\n",L,S,V) ;
}
```

运行结果如下：

```
L=31.415926
S=78.539815
V=523.598767
```

2. 带参宏定义

C 语言允许宏带有参数。在宏定义中的参数称为形式参数，在宏调用中的参数称为实际参数。对带参数的宏，在调用中，不仅要进行宏展开，而且要用实参去代换形参。

带参宏定义的一般形式为：

#define　宏名(形参表列)　字符串

在字符串中含有各个形参。

带参宏调用的一般形式为：

宏名(实参表列);

例如：

```
#define  M(y)  y*y+3*y      /*宏定义*/
……
k=M(5);                  /*宏调用*/
……
```

在宏调用时，用实参 5 去代替形参 y，经预处理宏展开后的语句为：

```
k=5*5+3*5;
```

【例 6.18】 求两数中较大者。

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

```
#define MAX(a,b) (a>b)?a:b
void main()
{
   int x,y,mymax;
   printf("请输入 x,y 的值:\n");
   scanf("%d,%d",&x,&y);
   mymax=MAX(x,y);
   printf("max=%d\n",mymax);
}
```

上例程序的第二行进行带参宏定义，用宏名 MAX 表示条件表达式(a>b)?a:b，形参 a，b 均出现在条件表达式中。程序第 8 行 mymax=MAX(x,y)为宏调用，实参 x，y 将代换形参 a，b。宏展开后该语句为：

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

用于计算 x，y 中的较大者。

对于带参的宏定义有以下问题需要说明：

（1）带参宏定义中，宏名和形参表之间不能有空格出现。

例如，把

```
#define MAX(a,b) (a>b)?a:b
```

写为：

```
#define MAX(a,b) (a>b)?a:b
```

将被认为是无参宏定义，宏名 MAX 代表字符串 (a,b) (a>b)?a:b。宏展开时，宏调用语句：

```
mymax=MAX(x,y);
```

将变为：

```
mymax=(a,b)(a>b)?a:b(x,y);
```

这显然是错误的。

（2）在带参宏定义中，形式参数不分配内存单元，因此不必作类型定义。而宏调用中的实参有具体的值，要用它们去替换形参，因此必须作类型说明，这与函数中的情况是不同的。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，只是符号代换，不存在值传递的问题。

（3）在宏定义中的形参是标识符，而宏调用中的实参可以是表达式。

例如：

```
#define SQ(y) (y)*(y)
  ……
sq=SQ(a+1);
  ……
```

宏定义中形参为 y，宏调用时实参为 a+1，是一个表达式，在宏展开时，用 a+1 代换 y，再用(y)*(y) 代换 SQ，得到如下语句：

```
sq=(a+1)*(a+1);
```

这与函数的调用是不同的，函数调用时要把实参表达式的值求出来再传递给形参。而宏代换中对实参表达式不作计算直接照原样代换。

（4）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。

【例6.19】 计算下列函数值：

$$f(x) = x^3 + (x+1)^3$$

其中自变量 x 的值从键盘输入。

如果将计算 $x^3$ 的值定义为一个带参数的宏，即

```
#define F(x) x*x*x
```

计算函数值 f(x) 的 C 程序写成如下：

```
#include<stdio.h>
#define F(x) x*x*x
main()
{  double f,x;
   printf("input x:");
   scanf("%lf",&x);
   f=F(x)+F(x+1);
   printf("f=%f\n",f);
}
```

运行时，当输入 1 后，结果为 5.0，而不是期望的 9.0。因为当 C 编译系统编译预处理 F(x+1) 时，经宏展开后，等价于“x+1*x+1*x+1”，而不等价于“(x+1)*(x+1)*(x+1)”。因此，为了使定义的宏展开合理，就必须将宏定义字符串中的参数都用括号括起来，即将上述程序改为：

```
#include<stdio.h>
#define F(x) (x)*(x)*(x)
main()
{  double f,x;
   printf("input x:");
   scanf("%lf",&x);
   f=F(x)+F(x+1);
   printf("f=%f\n",f);
}
```

经编译预处理后等价于下面的程序：

```
#include<stdio.h>
main()
{  double f,x;
   printf("input x:");
   scanf("%lf",&x);
   f=(x)*(x)*(x)+(x+1)*(x+1)*(x+1);;
   printf("f=%f\n",f);
}
```

在使用带参数的宏定义时，除了应将宏定义字符串中的参数都要用括号括起来，还需要将整个字符串部分也用括号括起来，否则经过宏展开后，还可能出现意想不到的错误。下面的例子说明了这个问题。

【例 6.20】 计算下列函数值：

$$f(x,y)=(x^3+x^2)[(y+1)^3+(y+1)^2]$$

其中子变量 x 的值从键盘输入。

如果将计算 $x^3+x^2$ 的值定义为一个带参数的宏，即

```
#define F(x) (x)*(x)*(x)+(x)*(x)
```

此时，在程序中就可以将 $x^3+x^2$ 写成 F(x)，将 $(y+1)^3+(y+1)^2$ 写成 F(x+1)。计算函数 f(x,y) 的 C 程序就可以写成：

```
#include<stdio.h>
#define F(x) (x)*(x)*(x)+(x)*(x)
main()
{   double f,x,y;
    printf("input x,y:");
    scanf("%lf,%lf",&x,&y);   /*输入的两个数之间用逗号做分隔符*/
```

```
    f=F(x)*F(y+1);
    printf("f=%f\n",f);
}
```

这个程序经编译预处理宏展开后 ，赋值语句：

```
f=F(x)*F(y+1);
```

等价于语句：

$$f = x^3 + x^2 * (y+1)^3 + (y+1)^2$$

这显然是错误的。正确的应该是：

$$f = (x^3 + x^2) * ((y+1)^3 + (y+1)^2)$$

有这个例子可以看出，为了使定义的宏展开合理，不仅需要将宏定义字符串中的参数都括起来，还必须将整个字符串用括号括起来。即将上述程序修改为：

```
#include<stdio.h>
#define F(x) ((x)*(x)*(x)+(x)*(x))
main()
{   double f,x,y;
    printf("input x,y:");
    scanf("%lf,%lf",&x,&y);/*输入的两个数之间用逗号做分隔符*/
    f=F(x)*F(y+1);
    printf("f=%f\n",f);
}
```

此时上面的程序等价于下面的程序：

```
#include<stdio.h>
main()
{  double f,x,y;
   printf("input x,y:");
   scanf("%lf,%lf",&x,&y);/*输入的两个数之间用逗号做分隔符*/
   f=(x3+x2)*((y+1)3+(y+1)2);
   printf("f=%f\n",f);
}
```

有上面的分析可以看出，在使用带参数的宏定义时，一般应将宏定义字符串中的参数都用括号括起来，并且整个字符串部分也要用括号括起来，这样才能保证在任何替代情况下，把宏定义作为一个整体来看待，从而得到一个合理的计算顺序，否则经过宏展开后可能出现意想不到的错误。

在 C 程序中，可以利用带参数的宏定义来表示一些比较简单的函数表达式。

（5）带参的宏和带参函数很相似，但有本质上的不同，除上面已讲到的几点外，把同一表达式用函数处理与用宏处理两者的结果有可能是不同的。

【例 6.21】 用用户自定义函数求 $n^2$。

```
#include"stdio.h"
void main()
{
   int i=1;
   while (i<=5)
     printf("%d\n",sq(i++));
}
int sq(int y)
{
    return((y)*(y));
}
```

运行结果如下：

```
1
4
9
16
25
```

【例 6.22】 用宏定义函数求 $n^2$。

```
#include"stdio.h"
#define SQ(y) ((y)*(y))
void main()
{
   int i=1;
   while (i<=5)
      printf("%d\n",SQ(i++));
}
```

运行结果如下：

```
2
12
30
```

在例 6.18 中函数名为 sq，形参为 y，函数体表达式为((y)*(y))。在例 6.19 中宏名为 SQ，形参也为 y，字符串表达式为((y)*(y))。例 6.18 的函数调用为 sq(i++)，例 6.19 的宏调用为 SQ(i++)，实参也是相同的。从输出结果来看，却大不相同。

分析：在例 6.18 中，函数调用是把实参 i 值传给形参 y 后自增 1，然后输出函数值。因而要循环 5 次，输出 1～5 的平方值。而在例 6.19 中宏调用时，只作代换。SQ(i++)被代换为((i++)*(i++))。在第一次循环时，由于 i 等于 1，其计算过程为：表达式中前一个 i 初值为 1，然后 i 自增 1 变为 2，因此表达式中第 2 个 i 初值为 2，相乘的结果也为 2，然后 i 值再自增 1 得 3。在第二次循环时，i 值已有初值为 3，因此表达式中前一个 i 为 3，后一个 i 为 4，乘积为 12，然后 i 再自增 1 变为 5。进入第三次循环，由于 i 值已为 5，所以这将是最后一次循环。计算表达式的值为 5*6 等于 30。i 值再自增 1 变为 6，不再满足循环条件，停止循环。

从以上分析可以看出，函数调用和宏调用二者在形式上相似，但在本质上是完全不同的。宏定义只是将字符串定义为宏名，其目的是为了避免重复的代码输入工作。虽然利用宏定义可以实现函数功能，计算表达式的值，但并不能像函数那样直接实现计算并带回返回值。在实际应用中，根据具体情况，如果使用宏定义能够使得程序简化，那么就可以使用宏来实现函数的功能。

### 6.6.2 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。文件包含命令行的一般形式为：

```
#include"文件名"
```

前面已多次用此命令包含过库函数的头文件，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。

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，由多个程序员分别编程。有些公用的符号常量或宏定义等可单独组成一个文件，在其他文件的开头用包含命

令包含该文件即可使用。这样，可避免在每个文件开头都去书写那些公用量，从而节省时间，并减少出错。

对文件包含命令还要说明以下几点：

（1）文件包含命令中的文件名可以用双引号括起来，也可以用尖括号括起来。例如以下写法都是允许的：

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

但是这两种形式是有区别的：使用尖括号表示在包含文件目录中查找（包含目录是由用户在设置环境时设置的），而不在源文件目录查找；使用双引号则表示首先在当前的源文件目录中查找，若未找到才到包含目录中查找。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。

（2）一个 include 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 include 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。

文件包含使得编程工作变得更加有效。C 语言也提供了许多库函数，包括标准函数和宏定义，存放在相关的后缀为“.h”的头文件中。当需要使用这些内容时，必须在源程序中用 #include 命令包含相关的头文件。而且用户也可以自己建立可供公用的头文件，在使用时将其包含在源文件中即可。

【例 6.23】 将求圆球体体积的函数作为头文件，并在其他程序中使用该函数。

将以下程序源代码保存到文件 sphere.h 中：

```
#define  PI  3.1415926
float volumn(float r)
{
   float v;
   v=4.0/3.0*PI*r*r*r;
   return v;
}
```

然后在需要求圆球体体积的程序中包含该文件，随后就可以使用该头文件中的内容了。

程序如下：

```
#include"sphere.h"
#include"stdio.h"
void main()
{
   float r,v;
   printf("r=");
   scanf("%f",&r);
   v=volumn(r);
   printf("PI=%f\nvolumn=%f\n",PI,v);
}
```

运行结果如下：

```
r=3
PI=3.1415926
volumn=113.097336
```

### 6.6.3　条件编译

预处理程序提供了条件编译的功能。可以按不同的条件去编译不同的程序部分，因而产

生不同的目标代码文件。这对于程序的移植和调试是很有用的。

下面分别介绍条件编译的三种形式。

（1）第一种形式：

#ifdef 标识符

程序段 1

#else

程序段 2

#endif

其中，标识符是指已用宏命令#define 定义的宏名，而程序段可以是编译预处理命令行，也可以是 C 语言组。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。如果没有程序段 2（它为空），本格式中的#else 可以没有，即可以写为：

#ifdef 标识符

程序段

#endif

【例 6.24】 条件编译命令的使用。

```
#include"stdio.h"
#define  REAL  float
void main()
{
   #ifdef  REAL
      REAL  a;
      printf("输入一个实数：");
      scanf("%f",&a);
      printf("这个实数是：%f\n",a);
   #else
      float  a;
      printf("输入一个单精度浮点数：");
      scanf("%f",&a);
      printf("这个单精度浮点数是：%f\n",a);
   #endif
 }
```

运行结果如下：

```
输入一个实数：4.56
这个实数是：4.560000
```

经过编译预处理后，上述程序变成如下形式：

```
void main()
{
      REAL  a;
      printf("输入一个实数：");
      scanf("%f",&a);
      printf("这个实数是：%f\n",a);
}
```

（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 表达式

程序段 1

#else

程序段 2

#endif

它的功能是：如表达式的值为“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。

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。

可以看到，使用条件编译可以提高编译的效率，并且可以使得程序更加灵活，有利于程序的调试以及移植等。

## 6.7　应用举例

【例 6.25】编写一个程序完成“菜单”功能，提供三种选择途径：其一是求水仙花数（所谓水仙花数是指三位整数的每一位上数字的立方和等于该整数本身。例如 153 就是一个水仙花数：$153=1^3+5^3+3^3$）。找出 100～999 之间的水仙花数。

其二是查找素数，找出 2～n 之间的素数。

其三是求 Faibonacci 数列前 n 项的值。

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
void main()
{  int m,xz;
   void narcissus();     /*声明求水仙花数的函数*/
   void prime();         /*声明查找素数的函数*/
   void faibonacci();    /*声明求 Faibonacci 数列前 n 项的函数*/
   system("cls");
   m=0;
   while(m==0)
     {  printf("\n");
        printf("1 求水仙花数\n");
        printf("2 查找素数\n");
        printf("3 求 Faibonacci 数列前 n 项\n");
        printf("输入了非法的数据,退出!\n");
        printf("\n");
        printf("请输入 1～3 中的一个数字：");
        scanf("%d",&xz);
```

```
        switch(xz)                              /*用开关语句 switch 进行选择*/
          { case 1:narcissus();break;           /*若选择 1，求水仙花数*/
            case 2:prime();break;               /*若选择 2，求素数*/
            case 3:faibonacci();break;          /*若选择 3,求 Faibonacci 数列前 n 项的值*/
            default:m=1;                        /*若选择其他讲 m 置 1，循环结束*/
      }
     }
 }
void narcissus()                                /*此函数用于求水仙花数*/
    {  int k,a,b,c,d;
       for(k=100;k<1000;k++)
       { a=k/100;
         b=k/100/10;
         c=k%10;
         d=a*a*a+b*b*b+c*c*c;
       if(d==k)
         printf("%d\n",k);
    }
 }
void prime()                                    /*此函数用于查找素数*/
{   int i,j,k,n,m=0;
   printf("请输入一个数:");
   scanf("%d",&n);
   for(i=2;i<=n;i++)
     {   j=sqrt(i);
         for(k=2;k<=j;k++)
           if(i%k==0)
              break;
         if(k>j)
          {  m++;
             printf("%3d",i);
             if(m%10==0)
             printf("\n");
          }
     }
}
void faibonacci()                       /*此函数用于求 Faibonacci 数列前 n 项的值*/
{    long int f,f1=1,f2=1;
     int i,n;
     printf("请输入 n: ");
     scanf("%d",&n);
     printf("%8ld%8ld",f1,f2);
     for(i=3;i<=n;i++)
        {
          f=f1+f2;
          f1=f2;                            /*存放第一个数*/
          f2=f;                             /*存放第二个数*/
          printf ("%8ld",f);
          if(i%6==0)                        /*为打印整齐，一行打印 6 个数，满 6 个数换行*/
             printf("\n");
       }
}
```

程序共有 4 个函数，其中主函数提供了主菜单，允许选择三种情况之一，否则就退出。

方法是：先输入选择，然后通过开关语句 switch 进行选择。为了不断地提供菜单，用 while 来循环。一开始给变量 m 赋初值为 0，m=0 就继续循环，一旦选择了不存在的情况，则将 m 置为 1，循环就结束，这是一种较为巧妙的程序设计技巧。

【例 6.26】 编写程序，解决万年历问题。

这里，编制的程序只要求解决公历问题，不需要考虑农历。整个程序的功能概括如下：

（1）输入一个年份，输出全年的日历。

（2）输入年份、月份，输出该月的日历。

（3）输入年月日，计算出这天是星期几。

这里将这个功能定义为一个功能模块，在主程序中分别调用，实现不同的功能。

在这个问题中，最主要是闰年的判断和如何确定每年的第一天是星期几。

某一年是闰年的条件为：该年份能被 4 整除但不能被 100 整除，或者能被 400 整除。根据这个条件，可以写出判断闰年的函数如下：

```
int isleap(int year)
{return (year%4==0&&year%100!=0||year%400==0);}
```

对于该函数，给它一个年份，如果返回值是 1 则说明是闰年，如果返回值是 0 则为非闰年。判断是否为闰年主要是为了确定 2 月份的天数，其他月份的天数是固定的。

确定一年的第一天是星期几的函数如下：

```
int day(int year)
{
long a,b;
if(year<=2000)       /*年份在2000年之前*/
   {
    a=2000-year;
    b=6-(a+a/4-a/100+a/400)%7;
    return b;
    }
else                 /*年份在2000年之后*/
   {
   a=year-2000;
   b=(a+(a-1)/4-(a-1)/100+(a-1)/400)%7;
   return b;
   }
}
```

在该函数中，对于年份首先判断是在 2000 年之前还是在 2000 之后，然后分别计算，最后返回该年份的第一天是星期几，这里，返回值 0～6 分别对应是星期天到星期六。

知道了每个月的天数和该年份的第一天是星期几，利用累加就能计算出每一天对应星期几，也就容易实现万年历系统的功能。

源程序如下：

（1）首先建立头文件 wnl.h，如下所示：

```
#include"stdio.h"
#define   D  "Sum  Mon  Tue  Wed  Thu  Fri  Sat"
void print(int n)           /*显示n个空格*/
{
int  i;
for(i=0;i<n;i++)
printf("  ");
}
```

```
int day (int year)          /*确定年份 year 的第一天是星期几*/
{
long  a,b;
if(year<=2000)              /*年份在 2000 年之前*/
   {
   a=2000-year;
   b=6-(a+a/4-a/100+a/400)%7;
   return b;
   }
else                        /*年份在 2000 年之后*/
   {
   a=year-2000;
   b=(a+(a-1)/4-(a-1)/100+(a-1)/400)%7;
   return b;
   }
}
int isleap(int year)        /*判断闰年*/
{
return(year%4==0&&year%100!=0||year%400==0);
}
```

在文件头中，包含了闰年的判断，确定一年的第一天是星期几，显示 n 个空格等函数，以及几个宏定义和必需的头文件 stdio.h。

（2）显示一年的日历，定义文件 year.c 如下：

```
void oneyear()
{
int a[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};   /*每个月的天数*/
int i,j,k,m,n,f1,f2,year,d;
clrscr();
printf("请输入年份：\n") ;
scanf("%d",&year);
printf("你所输入的年份是：%d\n",year);
d=day(year);    /*确定该年的第一天是星期几，用变量 d 表示*/
if(isleap(year))
  a[2]++;      /*如果是闰年则 2 月份的天数加 1*/
for(i=1;i<=12;i+=2)
{
m=n=f1=f2=0;
switch(i)       /*显示月份*/
{
case  1 :printf("January  1");break;
case  3 :printf("March   3");break;
case  5 :printf("May   5");break;
case  7 :printf("July  7");break;
case  9 :printf("September  9");break;
case  11:printf("November  11");break;
}
 print(21);
switch(i+1)       /*显示月份*/
{
case  2 :printf("February  2");break;
case  4 :printf("April  4");break;
case  6 :printf("June  6");break;
```

```
case  8 :printf("August  8");break;
case  10:printf("October  10");break;
case  12:printf("December  12");break;
}
printf("\n");
printf(D);print(6);printf(D);printf("\n");     /*显示日历的表头*/
for(j=0;j<6;j++)        /*显示每一天的日期*/
{
if(j==0)                /*第一行*/
{
print(d*4) ;
for(k=0;k<7-d;k++)
printf("%4d",++m);
print(6);
d=d+a[i]%7;
d%=7;                   /*确定下个月的第一天是星期几*/
print(d*4);
for(k=0;k<7-d;k++)
printf("%4d",++n);
printf("\n");
}
else     /*其他行*/
{for(k=0;k<7;k++)
  {if (m<a[i])
     printf("%4d",++m);
     print(4);
     if(m==a[i])
     f1=1;    /*该月显示完毕*/
          }
print(6);
for(k=0;k<7;k++)
{if(n<a[i+1])
  printf("%4d",++n);
  else
     print(4);
     if(n==a[i+1])
     f2=1;                  /*该月显示完毕*/
}
     printf("\n");
if(f1&&f2)
break;                      /*两个月显示完毕*/
}
}
d=d+a[i+1]%7;d%=7;          /*确定下个月的第一天是星期几*/
  printf ("");
for(k=0;k<27;k++)
    printf("=");            /*显示分隔符*/
    print(7);
    for(k=0;k<27;k++)
    printf("=");            /*显示分隔符*/
    printf("\n");
```

```
   if(i==5)
     {getch();
      clrscr();}
}
getch();
}
```

该函数将每年的日历分两屏显示，每屏显示 6 个月，在这个函数中，变量 m，n 分别为同一行上两个月的日期，利用累加实现。变量 d 用来记录每个月第一天是星期几，通过它确定每个月显示的位置。

（3）显示每个月的日历，定义文件 month.c 如下：

```
void onemonth()
{
   int a[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};   /*每个月的天数*/
   int i,j,k，m,flag,year,month,d;
   clrscr();                                            /*清屏*/
   printf("请你输入年份和月份");
   scanf("%d%d",&year,&month);                          /*输入年份和月份*/
   printf("\nThe calendar of %d\n",year);               /*输出表头*/
switch(month)
{  case  1 :printf("January  1");break;
   case  2 :printf("February  2");break;
   case  3 :printf("March   3");break;
   case  4 :printf("April  4");break;
   case  5 :printf("May   5");break;
   case  6 :printf("June  6");break;
   case  7 :printf("July  7");break;
   case  8 :printf("August  8");break;
   case  9 :printf("September  9");break;
   case  10:printf("October  10");break;
   case  11:printf("November  11");break;
   case  12:printf("December  12");break;
}
printf(D);
printf("\n\n");
 d=day(year);                  /*确定该年的第一天是星期几*/
if(isleap(year))
  a[2]++;                      /*如果是闰年，则 2 月的天数加 1*/
  for(i=1;i<month;i++)         /*确定该月第一天是星期几*/
    {d+=a[i]%7;d%=7;}
    m=flag=0;
     for(i=0;i<6;i++)          /*显示该月的日历*/
       {
        if(i==0)               /*显示第一行*/
         {
          print(4*d);
          for(j=0;j<7-d;j++)
           printf("%4d",++m);
           printf("\n");
          }
        else
         {
         for(j=0;j<7;j++)
```

```
        if(m<a[month])
        printf("%4d",++m);
        else
         flag=1;
         printf("\n");
         if(flag)
           break;
          }
        }
printf(" ");
for(k=0;k<27;k++)           /*分隔符*/
    printf("=");
    printf("\n");
    getch();
}
```

（4）输入年月日，计算出这天是星期几，定义文件 day.c 如下：

```
void oneday()
   {
   int a[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};  /*每个月的天数*/
   int i,year,month,dday,d;
   clrscr();                        /*清屏*/
   printf("请你输入年月日");
   scanf("%d%d%d",&year,&month,&dday);   /*输入年月日*/
   printf("你输入的是年月日是%d 年%d 月%d 日", year,month,dday);
   d=day(year);                     /*确定该年的第一天的星期天数目*/
   if(isleap(year))
        a[2]++;                     /*如果是闰年，则 2 月份的天数加 1*/
      for(i=1;i<month;i++)          /*确定该月第一天的星期数*/
        {d+=a[i]%7;d%=7;}
        d+=(dday-1)%7;d%=7;         /*确定该天的星期数*/
      switch(d)
        {
        case  0: printf("Sunday.\n");break;
        case  1: printf("Monday.\n");break;
        case  2: printf("Tuesday.\n");break;
        case  3: printf("Wednesday.\n");break;
        case  4: printf("Thursday.\n");break;
        case  5: printf("Friday.\n");break;
        case  6: printf("Saturday.\n");break;
        }
        getch();
   }
```

该函数中首先要确定该年份第一天的星期数，而后计算对应月份第一天的星期数，再计算出该天的星期数，最后输出结果。

（5）主程序。定义文件 wnl.c 如下：

```
#include"year.c"
#include"month.c"
#include"day.c"
void  main()
{
int choice,flag;
while(1)
```

```
{
flag=0;              /*标志变量赋初值*/
clrscr();            /*清屏，以下是显示功能菜单*/
printf("************************\n");
printf("*Please select  the  function(0--3):*\n");
printf("*1 Dispiay the calendar of one year.*\n");
printf("*2 Dispiay the calendar of one month.*\n");
printf("*3 Display the weekday of one day.*\n");
printf("*0.Exit*\n”);
printf("* **********************\n");
scanf("%d",&choice);
  switch(choice)
{
case  0:
        clrscr ();
        printf ("Thank you  for use this software.\n");  /*显示信息*/
        printf("Welcome to use it again\n");
        flag=1;break;
case  1:oneyear();break;             /*显示全年*/
case  2:onemonth();break;            /*显示某个月*/
case  3:oneday();break;              /*确定某一天的星期数*/
default:printf("Your choice wrong,please input again.");
        getch();
        }               /*结束 switch   */
        if(flag)
            break;
            }           /*结束 while*/
}                       /*结束  main */
```

程序说明：

（1）该程序包含所有子函数的源文件，根据用户选择的功能调用相应的自定义子函数。

（2）在主程序中，菜单循环显示，用户可以一直使用，直到选择功能键“0”，退出程序。

# 习　　题

## 一、选择题

1．以下说法中正确的是（　　）。

A．C 语言程序总是从第一个函数开始执行

B．在 C 语言程序中，要调用的函数必须在 main()函数中定义

C．C 语言程序总是从 main()函数开始执行

D．C 语言程序中的 main()函数必须放在程序的开始部分

2．某文件中定义的静态全局变量（或称静态外部变量）其作用域是（　　）。

A．只限某个函数　　B．本文件

C．跨文件　　D．不限制作用域

3．下列叙述中正确的是（　　）。

A．函数定义不能嵌套，但函数调用可以嵌套

B．函数定义可以嵌套，但函数调用不可以嵌套
C．函数定义和函数调用都不能嵌套
D．函数定义与函数调用都可以嵌套

4．以下说法中正确的是（　　）。
A．#define 和 printf 都是 C 语句　　B．#define 是 C 语句，而 printf 不是
C．printf 是 C 语句，但#define 不是　　D．#define 和 printf 都不是 C 语句

5．有以下函数：
```
char *fun(char *p)
{ return p; }
```
该函数的返回值是（　　）。
A．无确切的值　　B．形参 p 中存放的地址值
C．一个临时存储单元的地址　　D．形参 p 自身的地址值

6．以下是一个自定义函数的头部，其中正确的是（　　）。
A．int fun(int a[ ],b)　　B．int fun(int a[ ],int a)
C．int fun(int *a,int b)　　D．int fun(char a[ ][ ],int b)

7．以下关于函数的叙述中，正确的是（　　）。
A．在函数体中可以直接引用另一个函数中声明为 static 类别的局部变量的值
B．在函数体中至少必须有一个 return 语句
C．在函数体中可以定义另一个函数
D．在函数体中可以调用函数自身

8．在 C 语言中，若对函数类型未加显式说明，则函数的隐含类型是（　　）。
A．void　　B．double　　C．int　　D．char

9．下面不正确的描述为（　　）。
A．调用函数时，实参可以是表达式
B．调用函数时，实参与形参可以共用内存单元
C．调用函数时，将为形参分配内存单元
D．调用函数时，实参与形参的类型必须一致

10．有关宏定义的正确说明是（　　）。
A．可出现在一行中的任何位置
B．只能放在程序的开头，且每一个宏定义单独占一行
C．可出现在程序的任何位置
D．以#开头的行，可出现在程序的任何位置，通常每一个宏定义只能单独占一行，使用字符“\”可实现一个宏定义占用若干行

11．若程序中有宏定义行：
```
#define N 100
```
则以下叙述中正确的是（　　）。
A．宏定义行中定义了标识符 N 的值为整数 100
B．在编译程序时对 C 源程序进行预处理时用 100 替换标识符 N
C．对 C 源程序进行编译时用 100 替换标识符 N
D．在运行时用 100 替换标识符

12．若调用函数的实参为变量时，以下关于函数形参和实参的叙述中正确的是（　　）

A．函数的实参和其对应的形参共占同一存储单元
B．形参只是形式上的存在，不占用具体存储单元
C．同名的实参和形参占同一存储单元
D．函数的形参和实参分别占用不同的存储单元

13．C 语言规定，函数返回值的类型是由（　　）。
A．return 语句中的表达式类型所决定
B．调用该函数时的主调函数类型所决定
C．调用该函数时系统临时决定
D．在定义该函数时所指定的函数类型所决定

14．关于函数的定义和调用，以下说法正确的是（　　）。
A．用户若需调用标准库函数，调用前必须重新定义
B．用户可以重新定义标准库函数，该函数将失去原有含义
C．系统根本不允许用户重新定义标准库函数
D．用户若需调用标准库函数，调用前不必使用预编译命令将该函数所在文件包括到用户源文件中，系统自动去调用

15．C 语言关于实参类型的规定，以下说法不正确的是（　　）。
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．实参可以为任何类型
D．形参应与其对应的实参类型一致

16．C 语言规定：简单变量做实参时，它和对应形参之间的数据传递方式是（　　）。
A．由用户指定的传递方式
B．单向值传递
C．由实参传给形参，再由形参传回给实参
D．地址传递

17．以下关于实参和形参所占存储单元的叙述中说法正确的是（　　）。
A．在 C 语言中实参和与其对应的形参各占用独立的存储单元
B．在 C 语言中实参和与其对应的形参共占用一个存储单元
C．在 C 语言中只有当实参和与其对应的形参同名时才共占用存储单元
D．在 C 语言中形参是虚拟的，不占用存储单元

18．以下对 C 语言函数的有关描述中，正确的是（　　）。
A．调用函数时，只能将实参的值传送给形参，形参的值不能传送给实参
B．C 函数既可以嵌套定义又可以递归调用
C．函数必须有返回值，否则不能使用函数
D．C 程序中有调用关系的所有函数必须放在同一个源程序文件中

19．若调用一个函数，且此函数中没有 return 语句，则该函数（　　）。
A．没有返回值　　B．返回若干个系统默认值
C．能返回一个用户所希望的值　　D．返回一个不确定的值

20．若已定义的函数有返回值，则以下关于该函数调用的叙述中错误的是（　　）。
A．函数调用可以作为独立的语句存在
B．函数调用可以作为另一个函数的实参

C．函数调用可以出现在表达式中

D．函数调用可以作为另一个函数的形参

21．以下函数调用语句的实参个数为（　　）。

```
fun((ex1,exp2),(exp3,exp4,exp5));
```

A．1　　B．2　　C．4　　D．5

22．以下函数值的类型是（　　）。

```
fun(float x)
{ float y;
  y=3*x-4;
  return y;
}
```

A．int　　B．不确定　　C．void　　D．float

23．若有以下程序，则以下叙述中不正确的是（　　）

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

A．若只在主函数中对函数 f 进行说明，则只能在主函数中正确调用函数 f

B．若在主函数前对函数 f 进行说明，则在主函数和其后的其他函数中都可以正确调用函数 f

C．对于以上程序，编译时系统会提示出错信息，提示对函数 f 重复说明

D．函数 f 无返回值，所以可用 void 将其类型定义为无值型

24．一个 C 语言源程序文件中所定义的全局变量的作用域为（　　）。

A．所在文件的全部范围

B．所在程序的全部范围

C．所在函数的全部范围

D．由具体定义位置和 exlern 说明来决定范围

25．在 C 语言中，存储类型为（　　）的变量只有在使用它们时才占存储空间。

A．static 和 auto　　B．register 和 auto

C．static 和 register　　D．register 和 extern

26．若一个外部变量的定义形式为 static int x；则 static 的作用应当是（　　）。

A．将变量存放在静态存储区　　B．使变量 X 可以由系统初始化

C．使变量 X 只能在本文件内使用　　D．使 X 的值可以永久保留

## 二、填空题

1．编译预处理的三种形式除宏定义外，还有_____和_____。

2．C 语言规定，调用一个函数时，实参变量和形参变量之间的数据传递方式是_____。

3．C 语言变量按其作用域可分为_____和_____，按其生存期可分为_____和_____。

4．C 语言变量的存储类别有_____、_____、_____和_____。

## 三、阅读程序，写出执行结果

1．以下程序的运行结果是（　　）。

```
int w=0;
void fun()
{
   printf("%d,",w);
}
void main()
{
   int w=5;
   printf("%d,",w);
   fun();
   printf("%d,",w);
}
```

2．以下程序的运行结果是（　　）。

```
int k=1;
void fun(int n)
{
           m+=k;
           k+=m;
           printf("%d,5d",m,k);
}
  int j=2;
 main()
       {
          int i=3;
          fun(i);
          printf("%d,%d,%d\n",i,j,k);
    }
```

3．以下程序的运行结果是（　　）。

```
#define F(x)  4*x*x+1
main()
  {   int i=1,j=3;
      printf("%d,%d",F(5)*2,F(i+j)*2);
  }
```

4．以下程序的运行结果是（　　）。

```
#include<stdio.h>
fun(int x,int ,y,int z)
{   z=x*x+y*y;
    returnz;
}
void main()
{ int a=31;
   fun(5,2,a);
   printf("%d",a);
}
```

5．以下程序的运行结果是（　　）。

```
#include<stdio.h>
float fun(int x,int y)
{    return(x+y); }
```

```
void main()
{  int a=2,b=3,c=8;
   printf("%3.0f\n",fun((int)fun(a+c,b),a-c));
}
```

6. 以下程序的运行结果是（　　）。

```
#include<stdio.h>
int f(int n)
{   if(n==1) return 1;
   else return f(n-1)+1;
 }
 void main()
 {   int i,j=0;
   for(i=1;i<3;i++)
      j+=f(i);
    printf("%d\n",j);
}
```

7. 以下程序的运行结果是（　　）。

```
void main()
{
   int i,j,x,y,n,g;
   void fun(int i,int j);
   i=2;j=3;g=x=5;y=9;n=7;
   fun(n,6);
   printf("g=%d;i=%d;j=%d\n",g,i,j);
   printf("x=%d;y=%d\n",x,y);
   fun(n,6);
}
void fun(int i,int j)
{
   int x,y,g;
   g=8;x=7;y=2;
   printf("g=%d;i=%d;j=%d\n",g,i,j);
   printf("x=%d;y=%d\n",x,y);
   x=8;y=6;
}
```

8. 以下程序的运行结果是（　　）。

```
void main()
{
      int i,j;
      int ran();
      int rand();
      for(i=0;i<3;i++)
        {
              for(j=0;j<2;j++)
                 printf("%3d",ran());
              printf("\n");
         }
      for(i=0;i<3;i++)
       {
         for(j=0;j<2;j++)
```

```
            printf("%3d",rand());
          printf("\n");
       }
}
int ran()
{
   static int see=1234,n;
   printf("(see=%4d)”,see);
   see=(see+25543)%7415;
   n=(see%1000)/10;
   return(n);
}
int rand()
{
   int see=1234,n;
   printf("(see=%4d) ",see);
   see=(see+25543)%7415;
   n=(see%1000)/10;
   return(n);
}
```

9. 以下程序的运行结果是（　　）。

```
#include"stdio.h"
#define LOW 0
#define HIGH 5
#define CHANGE 2
int i=LOW;
void main()
{
     int workover(int i);
     int reset(int i);
     int i=HIGH;
     reset(i/2);
     printf("i=%d\n",i);
     reset(i=i/2);
     printf("i=%d\n",i);
     reset(i/2);
     printf("i=%d\n",i);
     workover(i);
     printf("i=%d\n",i);
 }
int workover(int i)
{
   i=(i%i)*((i*i)/(2*i)+4);;
   printf("i=%d\n",i);
   return(i);
}
int reset(int i)
{
   i=i<=CHANGE?HIGH:LOW;
   return(i);
}
```

## 四、编程题

1．写两个函数，分别求两个整数的最大公约数和最小公倍数，用主函数调用这两个函数，并输出结果，两个整数由键盘输入。

2．采用函数的调用，编写程序求三个数中的最大数。

3．设计一个求 100～200 间所有素数之和的程序，其中素数的判定用函数实现。

4．采用函数调用的方法，编程求梯形的面积（上底、下底和高的值从键盘输入）。

5．利用递归函数调用方式，将所输入的 5 个字符以相反顺序打印出来。