由 C 程序结构所知，一个完整的 C 语言程序是由一个且只能有一个 main() 函数 (又称主函数) 和若干个其他函数组合而成的。而前面各章仅学习 main() 函数的编程，本章将介绍其他函数的编程，包括其他函数的定义、调用、参数传递及变量的作用域等。

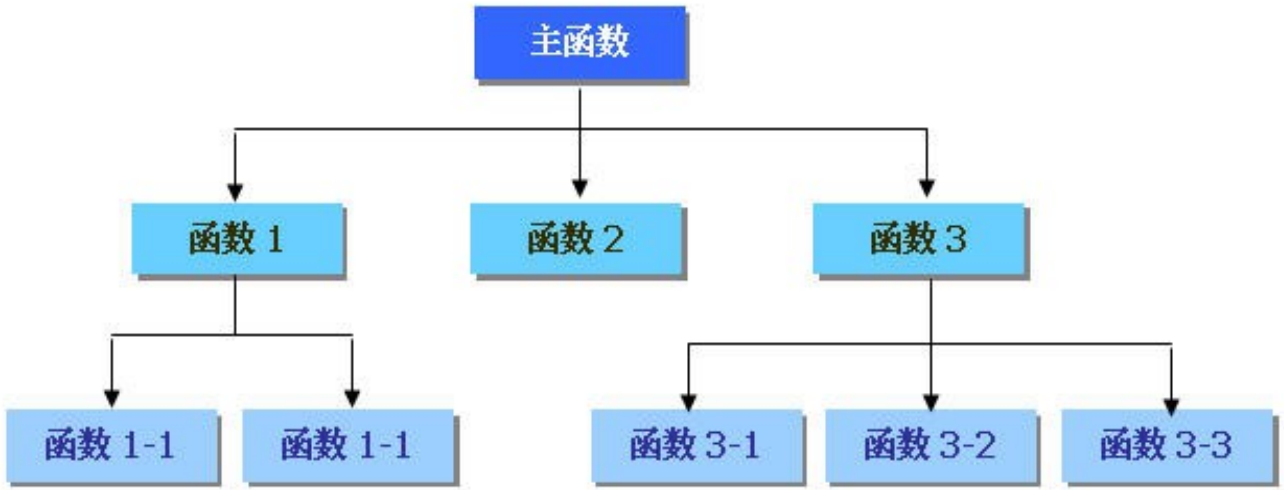


图 7.1 C 语言程序结构

第 7 章　函　数

7.1　函数的定义和调用

7.2　变量的作用域

7.3　变量的存储类型

7.4　函数间的数据传送

7.5　函数的嵌套调用和递归调用

7.6　内部函数和外部函数

7.7　程序综合举例

7.1 函数的定义和调用

7.1.1 函数的定义

❖ 函数定义的一般形式：（有如下两种）

1．函数定义的传统形式

< 存储类型 > < 数据类型 > < 函数名 > （ < 形参表 > ）
形参类型说明语句序列；
{
　　< 函数体 >
}

2．函数定义的现代风格形式

< 存储类型 > < 数据类型 >< 函数名 > （ < 类型 参数 1>
，< 类型 参数 2>，……）
{
　　< 函数体 >
}

例如：一个求和函数

可以写成：

```
int sum(x,y)
int x;
int y;
{
return(x+y);
}
```

也可写成：

```
int sum(int x, int y)
{
return(x+y);
}
```

❖ 关于函数定义的几点说明：

(1) 一个源程序文件仅一个 main 的函数，并从 main 函数开始执行，调用其他函数后流程回到 main( ) 函数，在 main( ) 函数结尾结束整个程序的运行。

(2) 一个 C 程序由一个或多个源程序文件组成。

(3) 函数类型是该函数返回值的类型。有 int 、 float 、 char 等，若函数无返回值函数定义为空类型 void 。缺省为 int 。

(4) 函数名取用与标识相同。最好见名知意，以增强程序的可读性。

(5) 函数可为无参函数。但函数名后的（ ）不能省略。

(6) 函数参数为多个参数时，其间用逗号隔开。在调用时，主调函数将数据传送给被调用函数使用。

7.1.2 函数说明与调用

- 主调函数调用被调函数时，在调用前应先对被调函数进行说明，即先说明后调用。
- 函数说明的一般格式：

  < 存储类型 > < 数据类型 > < 函数名 > （ ）；

  说明：当被调函数定义在主调函数之前时，可以省略对被调函数的说明。

❖ 函数调用方式：

1．函数表达式

函数调用出现在一个表达式中，这种表达式称为函数表达式。调用后函数返回一个确定的值。

【例 7．1】求三个任意数中的最大数。

```
float  f(x,y)
float x, y;
{ float max;
max=x>y? x ：  y;
return(max);
}
```

```
#include "stdio.h"
void  main( )
{ float a,b,c,max;
printf(" 请输入任意三个实数：  ") ;
scanf("%f,%f, %f",&a,&b,&c);
max=f(a,b);
max=f(max,c) ;
printf(" 最大数是 :%f",max);
printf("\n");
 }
```

说明：因被调函数在主调函数前，程序中省略了对被调函数 f( ) 的说明。

2．函数参数

把函数调用作为一个函数的实在参数。

例如：

```
void  main( )
{ float a,b,c,max;
printf(" 请输入任意三个实数：  ") ;
scanf("%f,%f, %f",&a,&b,&c);
max=f(f(a,b),c);   /* 调用例 7.1 题的被调函数 */
printf(" 最大数是 :%f",max);
printf("\n");
 }
```

3．函数语句

把函数调用作为一个语句，不需要返回值。

例如：

```
#include "stdio.h"
void  main( )
{ float a,b,c,max;
void f( );                          /* 函数说明 */
printf(" 请输入任意三个实数：  ") ;
scanf("%f,%f, %f",&a,&b,&c);
f (a,b,c);                          /* 函数语句 */
printf("\n");
  }
```

```
void  f(x,y,z)
float x, y, z;
{ float max;
if (x>y) max=x;
else max=y;
    if (max<z) max=z;
printf(" 最大数是 :%f",max);
}
```

7.1.3 函数的返回值

函数调用后使主调函数能得到一个确定的值，这就是函数的返回值。

说明：

(1) 函数的返回值：是通 int 过函数中的 return 语句获得的。

(2) 函数类型：即为函数返回值的类型。

(3) 函数空类型：被调函数中没有 return 语句，函数返回一个不定值。若明确不需返回值，则用“ void”说明函数（或称“空类型”）。

7.2 变量的作用域

7.2.1 局部变量

局部变量：在一个函数内部定义的变量称为局部变量。局部变量只在定义的函数范围内有效，即才能使用它们。

例如：

```
float f1(int  a ，  float x)
{ int  b, c;            /* 变量 b 、 c 只能在 f1() 函数中有效 */
     …
}
char  f2(int  x, int  y)
{ float  a ;            /* 变量 a 只能在 f2 函数中有效 */
   …
}
void  main( )
{int m, n;              /* 变量 m 、 n 只能在主函数中有效 */
      …
}
```

7.2.2 全局变量

- 全局变量：在函数外部定义的变量称为全局变量，又称为外部变量。
- 全局变量可以为该文件中其它函数所共用。有效范围为从定义变量的位置开始到本源文件结束。

例如：

```
int  p,q ;              /* 定义全局变量 p,q */
float  f1(int a)
{ int  b, c;
      …
}
char  c1,c2;    /* 定义全局变量 c1,c2 */
char f2(int  x ,int y)
{floati, j;
   、
    …
   全局变量 c1        q 作用范围
void  main( )                          c2  作用范围
{ int m, n;
      …
}
```

全局变量 p

【例 7.2 】全局变量与局部变量同名。

```
int a=3,b=5;              /* 定义全局变量 a,b */
max(int a, int b)
{
   int  c;
   c=a>b?a:b;
   return(c);
}
void  main( )
{
   int a=8;
   printf("%d",max(a,b));
}
```

运行结果为：

8

说明：

（**1**）全局变量 a 、 b 在 max 函数内不起作用。

（**2**）全局变量 a 在 main 函数内不起作用（**因为**：全局变量与局部变量同名时，局部变量作用域内全局变量被“**屏蔽**”不起作用），而全局变量 b 在此范围内有效。

（**3**）利用全局变量增加与函数之间的联系，可以得到多个函数返回值。

【例 7.3 】一数组中存放有 10 个学生成绩，写一个函数求出平均分、最高分和最低分。

程序思想：被调函数需返回平均分、最高分和最低分三个数值，而 return() 只能返回一个值，所以将最高分 max 和最低分 min 设置为全局变量。

程序如下：

```
float max=0, min=0;          /* 定义全局变量 max,min */
float average (float a[ ], int n)  /* 定义求平均分、最高分和最低分函数  */
{
   int i;
   float aver, sum=a[0];
   max=min=a[0];
   for (i=1; i<n;i++)
```

```
{  if (a[i]>max) max=a[i];
     else  if (a[i]<min)  min=a[i];
   sum=sum+a[i];
}
aver=sum/n;
return(aver);
}
void  main(  )
{
   float ave,score[10];
   int  i;
   for(i=0; i<10; i++)
   scanf("%f",&score[i]);
   ave=average(score, 10);
   printf("max=%6.2f\nmin=%6.2f\naverage=%6.2f\n",ma
   x, min, ave);
}
```

运行结果为：

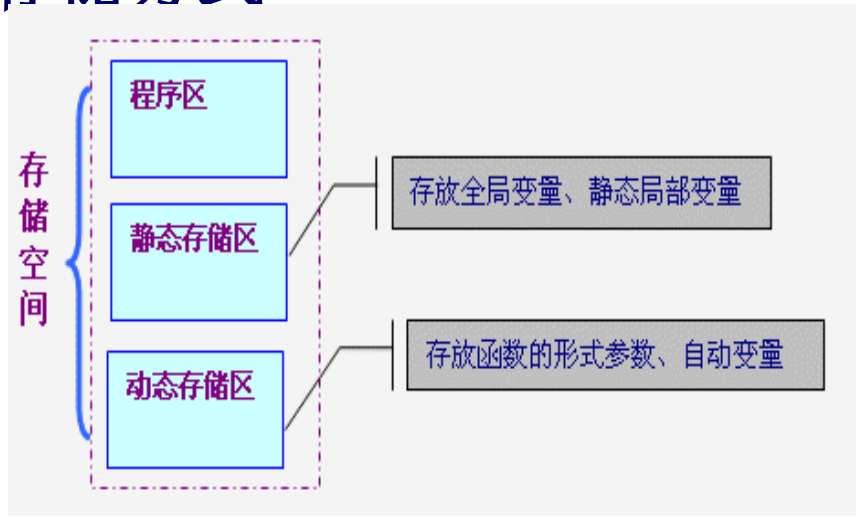
```
87  77  45  63  89  88  94  99  38  68↙
max=99.00
min=38.00
average=74.80
```

注：过多地使用全局变量，会降低程序的清晰度和通用性，难以清楚地判断出每个瞬时各个全局变量的值。因此建议不在必要时不要使用全局变量。

7.3 变量的存储类型

7.3.1 静态存储方式和动态存储方式

•变量值在内存的存在的时间来分为静态存储方式和动态存储方式。
•静态存储方式：在程序运行期间分配固定的存储空间；
•动态存储方式：在程序运行期间根据需要进行动态的分配存储空间。
•用户使用的存储空间：由程序区、静态存储区和动态存储区三部分组成（如下图所示）。



数据分别存放在静态存储区和动态存储区中。

动态存储区主要存放函数的形式参数、自动变量和函数用时的现场保护和返回地址等。在函数调用开始分配动态存储空间，函数结束时释放这些空间。

7.3.2　变量的存储类型

变量属性：数据类型属性和存储类型属性。

变量的存储类型： auto （自动）型、 static （静态）型、 register （寄存器）型和 extern （外部）型四种。

1 ． Auto(自动) 变量

auto 变量只用于定义局部变量，存储在内存中动态存储区。

定义形式为：

auto < 数据类型 > < 变量名表 > ;

局部变量存储类型缺省时为 auto 型。

例如：

```
int  f (int x)                  /* 定义 f 函数， a 为形参 */
{auto int a, b;                 /* 定义整型变量 a 、 b 为自动变量 */
  float  y;                     /* 定义 y ，缺省存储类型时为自动变量 */
…
}
```

2．static(静态) 变量

static 型可定义全局变量和局部变量。

static(静态) 变量在静态存储区分配存储单元。在程序运行行期间自始至终占用被分配的存储空间。

定义形式为：　注意：⑴静态局部变量是在编译时赋初值的。以后每次调用函数时不再重新赋初值而只引用上次函数调用结束时的值。

⑵ 静态局部变量时没有赋初值，编译时自动赋 0 或空字符（对字符变量）。

static < 数据类型 > < 变量名表 > ;

【例 7.4 】分析下列程序运行结果。

```
void  main( )
   { void f( );
     f( );
     f( );
     f( );
   }
void f( )
   {int x=0;
    x++;
    printf("%d\t",x);
   }
```

运行结果为：

1　　1　　1

3． register（寄存器）变量

- register（寄存器）变量：存放在寄存器中的变量。
- 变量的值一般是存放在内存中的。某些要频繁使用的变量，为了提高变量的存取时间，可将这些变量存放在寄存器中，将变量定义为 register 型。
- 定义形式为：

  register ＜数据类型＞ ＜变量名表＞；

注意：
⑴ 计算机中寄存器数量是有限的，因此不能太多的寄存器变量。
⑵ 只有局部自动变量和形式参数可以定为寄存器变量，全局变量和静态变量不能定义为寄存器变量。

【例 7.5 】分析下列程序存在的错误。

```
void  main( )
{
   register int  x;
   x=1000;
   printf("%d\n", &x);    /* 寄储器变量 x 不能使用 “ &” 运算符 */
}
```

注：寄储器变量没有存储器地址。

4．Extern(外部) 变量

- extern 变量就是全局变量。
- 全局变量是从作用域角度提出的；外部变量是从其存储方式提出的，表示它的生存存期。
- 外部变量的定义就是全局变量的定义。
- 若 extern 型变量的定义在后，使用在前，或者引用其它文件的 extern 型变量，这时必须用 extern 对该变量进行外部说明。
- 外部说明形式为：

  extern ＜数据类型＞ ＜变量名表＞；

【例 7.6 】 extern 型变量定义与外部说明示例。

```
#include "stdio.h"
   int b=3;                          /* 定义 extern 型变量 b*/
   void  main()
   {
    extern int  a;                   /*extern 型变量 a 的外部说明 *
   /
    printf("a=%d\tb=%d\n",a,b);
   }
   int a=18;                         /* 定义 extern 型变量 a*/
```

运行结果为：

a=18　　b=3

【例 7.7 】分析下列程序。

```
/* 源文件 file1.c*/
int i;
/* 定义 extern 型变量 i*/
void  main( )
  {
    i++;
    printf("i=%d\n",i);
    next( );
  }
  int i=3;  /*i 变量赋初值 */
  static int next( )
    {
      i++;
      printf("i=%d\n",i);
      other( );
    }
```

```
  /* 源文件 file2.c*/
    extern int i;
   /* 对 extern 型变量 i 进行外部说明 */
     int other( )
      {
        i++;
        printf("i=%d\n",i);
      }
```

运行结果为：

```
i=4
i=5
i=6
```

- 主调函数与被调函数之间的数据传递可通过参数进行，主调函数的参数称为实参，被调函数的参数称为形参。
- 函数间数据传递方式主要有：有传值方式、传址方式、利用参数返回结果、利用函数返回值和利用全局变量传递数据等。

7.4 函数间的数据传送

7.4.1　传值方式

传值方式：又称数据复制方式，就是把主调函数的实参值复制给被调用函数的形参，使形参获得初始值。

注意：

1．实参向形参传递数据是单向且一一对应。

2．形参、实参各占独立的存储空间。形参在函数被用时，动态分配临时存储空间，函数返回释放。因此实参与形参可以同名，也可以不同名，而且形参数值发生变化，实参值不变。

3．形参属于局部变量。

【例 7.8 】分析下列程序执行结果。

```
void  main()
  {    int a=3,b=5;
       void swap(int x,int y);                    /* 函数说明 */
       swap(a,b);
       printf("a=%d,b=%d\n",a,b);   /*a 、 b 变量为实参
  */
  }
void swap(int x, int y)                        /*x 、 y 变量为形参 *
  /
  {    int temp;
       temp=x,x=y,y=temp;
       printf("x=%d,y=%d\n",x,y);
  }
```

运行结果为:

```
x=5, y=3
a=3 , b=5
```

7.4.2　地址复制方式

- 地址复制方式：又叫传地址方式，就是把地址常量传送给形参。
- 地址传递方式中一般用数组名或指针作为形参接收实参数组首地址，使形参与实参地址相同。在被调函数中，如果修改了数组元素值，调用函数后实参数组元素值也发生相应变化。从而实现被函数返回多值给主调函数。

【例 7.9 】分析下列程序执行结果。

```
void  main()
{ static int a[3]={1,2,3};
  printf(" 调用函数前数组各元素值为：  ");
  printf("%d,%d,%d\n",a[0],a[1],a[2]);
  chg(a);
  printf(" 调用函数后数组各元素值为：  ");
  printf("%d,%d,%d\n",a[0],a[1],a[2]);
 }
```

```
chg(int b[ ])
  { int i;
    for (i=0;i<3;i++)
      b[i]=b[i]+1;
  }
```

运行结果为：
调用函数前数组各元素值为：
1， 2， 3
调用函数后数组各元素值为：
2， 3， 4

7.4.3　利用参数返回结果

使用地址复制方式，被调用函数修改主调函数中的数据，实现多值返回。

7.4.4　利用函数返回值传递数据

在被函数中用： return(表达式); 返回一个处理结果给主调函数。

7.4.5　利用全局变量传递数据

利用全局变量进行函数间的数据传递，简单而运行效率高。

全局变量使用过多增加了函数间联系的复杂性，降低了函数的独立性。

7.5 函数的嵌套调用和递归调用

7.5.1 函数嵌套调用

函数嵌套调用：在被调函数中再调用其它函数称函数嵌套调用。

◆ C 语言不能嵌套定义函数。

例如：在下列调用 f1 函数中调用 f2 函数。

```
float f1(int a, int b)
{   …
    f2(a+b,a-b);
    …
}
int f2(int x, int y)
{
    …
}
```

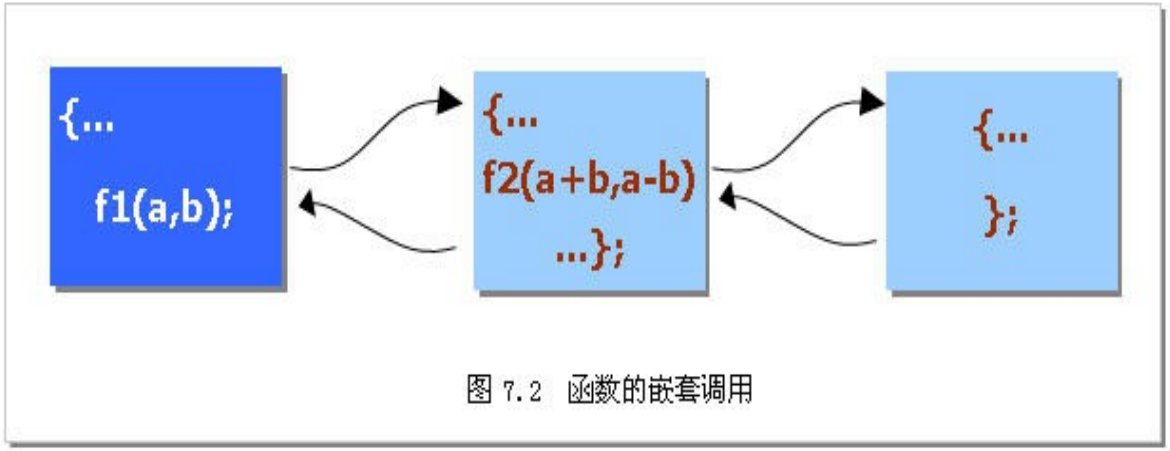


图 7.2　函数的嵌套调用

【例 7.10 】求 1k+2k+3k+…+nk 的值，假设 k 为 4 ， n 为 6 。

```
#include "stdio.h"
void main( )
{   int  sum, n=6, k=4;
    sum=add(k, n);
    printf(" 输出结果为：  %d",sum );
}
add(int a,int b)            /* 该函数功能：进行累加 */
{   int  i, s=0;
    for(i=1; i<=b; i++)
    s=s+powers(i,a);
    return(s);
}
```

```
powers(int m, int  n)        /* 该函数功能：进行累乘 */
{
   int  j, p=1;
   for (j=1; j<=n; j++)
   p=p*m;
   return(p);
}
```

运行结果为：

```
2275
```

7.5.2 函数递归调用

递归调用：在调用一个函数的过程中直接或间接地调用该函数本身，称为函数的递归调用。

在编写递归调用程序时注意：

(1) 递归程序算法：即如何实现其递归；

(2) 递归调用的结束条件：避免无止境递归调用造成死循环。所以递归调用应为条件递归调用：

```
if ( 条件 ) 递归调用
else  ……
```

【例 7.11 】用递归算法编程求 n! 阶乘的程序。

从数学可知： n!=1*2*3* … *n ， 可得： 1!=1

n!=n(n-1)!　当 n>1 时

递归算法：

递归调用的结束条件：{ 1!=1 ; n!=n(n-1)!　当 n>1 时

```
if( (n==0)|| (n==1))  return(1);
   else  return(n*fac(n-1));
 /* fac(n-1) 求 (n-1) ！ 函数 */
```

程序如下：

```
int  fac(int n)
{  if (n<0) printf("n<0 ， 输入数据错误！ ");
   else if( (n==0)|| (n==1))  return(1);
   else  return(n*fac(n-1));
}
void main( )
{  int  n, y;
   printf(" 请输入一个整数： ");
   scanf("%d",&n);
   y=fac(n);
   printf("%d!=%d",n, y);
}
```

运行结果为：

```
请输入一个整数：  5↙
5!=120
```

【例 7.12 】调用一个递归函数，将一个整数的低位变成高位，高位变成低位组成另一个整数，例如输入 1234 得到另一个整数 4321 。

```
#include "stdio.h"
int  fun(int  n, int m)
{
   if (n==0)  return m;
   else return fun(n/10, m*10+n%10);
}
void main(  )
{
   printf("%d\n",fun(1234, 0));
}
```

7.6 内部函数和外部函数

7.6.1 内部函数

内部函数：一个函数只能在所定义的源文件中进行函数调用，称此函数为内部函数。内部函数的存储类型为 static，又称为静态函数，

定义格式为：

static ＜类型标识符＞ ＜函数名＞（＜形式参数表＞）

【例 7.13 】分析下列程序。

```
/* 设该程序源文件名 file.c  */
void  main()
  {
    int i=2,j=3,p;
    extern int f();
    p=f(i,j);
    printf("%d",p);    }
```

```
static  int f(int a,int b)
    {
        int c;
        if(a>b) c=1;
        else if(a==b) c=0;
         else c=-1;
        return(c)
    }
```

static int f(int a,int b) 函数只能被文件 file.c 中的 main 函数调用，其它程序文件是不能调用该该函数的。

7.6.2 外部函数

- 若将函数的存储类型定义为 extern 型，则此函数能被其它源文件的函数调用，称此函数为外部函数。
- 外部函数的定格式为：
- extern < 类型标识符 > < 函数名 > （ < 形式参数表 > ）
- 定义函数时缺省存储类型为 extern ，即隐含为外部函数。
- 例如：将【例 7.13 】中 static int f(int a,int b) ；改为 extern int f(int a,int b) ；就可以被其它文件中的函数调用。

注意：在需要调用外部函数的文件中，一般要用 extern 进行函数的外部说明。

7.7 程序综合举例

【例 7.14 】编写一程序求三个数的最小公倍数。

程序思想：首先编写一函数求出三个数中的最大数；再用最大数依次乘以自然数 1 ， 2 ， 3 ， 4 ，……所得的积，分别去除以原三个数，满足都能除尽时的最小积，就是这三个数的最小公倍数。

程序如下：

```
#include <stdio.h>
void main()
  { int max(int x,int y,int z);
    int x1,x2,x3,k=1,j,x0;
scanf("%d,%d,%d",&x1,&x2 ,&
x3);
   x0=max(x1,x2,x3);
   while(1)
   {     j=x0*k;
         if(j%x1==0&&j%x2==0
&&j%x3==0 )
         break;
         k+=1;
    }
  printf("The result is %d\
    n",j);
}
  int max(int x,int y,int z )
    { if(x>y&&x>z)
        return x;
      else if(y>x&&y>z )
        return y;
      else
        return z;
    }
```

【例 7.15 】编写一个程序，判定 1+2+3+…+n 大于 1000 的最小整数 N 。

程序思想：编一函数 sum(n) 求自然数 1+2+3+…+n 的累加和。主函数中分别取实参为 1 ， 2 ， 3…… 去调用 sum(n) 函数，当所得结果 >1000 时，输出 n 值。

程序如下：

```
int  sum(int n)
  {  int total,i;
     total=0;
     for (i=1;i<=n;i++)
     total=total+i;
     return(total);
  }
```

```
void  main()
   { int n;
     n=1;
     while(sum(n)<=1000)
     n++;
     printf("1+2+3+...+n>1000 N 的极限值是 : %d",n);
  }
```

运行结果是：

1+2+3+...+n>1000 N 的极限值是： 45

【例 7.16 】有 6 个人坐在一起，问第 6 个人有多少岁，他说比第 5 个人大 3 岁；问第 5 个人有多少岁，他说比第 4 个人大 3 岁；问第 4 个人有多少岁，他说比第 3 个人大 3 岁；问第 3 个人有多少岁，他说比第 2 个人大 3 岁；问第 2 个人有多少岁，他说比第 1 个人大 3 岁；问最后个人，他说是 15 岁，请问第 6 个人多少岁？

程序思想：根据题意分析可得： age(1)=15

age(n)=age(n-1)+3 (n>1)

得递归算法为：

递归调用的结束条件： if(n==1)c=15;

else c=age(n-1)+3;

程序如下：

```
#include "stdio.h"
void age(n)
int n;
{
    int c;
    if(n==1)c=15;
    else c=age(n-1)+3;
    return(c);
}
void main(  )
{
     printf("%d",age(6));
}
```

程序的运行结果为：

30

【例 7.17 】编写一个程序，判断用户键入的月份有几天。

程序思想：首先编一函数 leapyear( ) 判断是否为闰年（闰年返回 1 ，不是返回 0 ）；在主函数中输入月份进行判断：

（ 1 ） 若输入月份为 1 、 3 、 5 、 7 、 8 、 10 或 12 ，则输出 31 天；

（ 2 ） 若输入月份为 4 、 6 、 9 或 11 ，则输出 30 天；

（ 3 ） 若输入月份为 2 ，调用函数 leapyear( ) 判断，若是闰年输出 29 天，否则输出 28 天。

程序如下：

```
#include "stdio.h"
 int leapyear(int year)
   { switch(year%4)
    { case 0:if(year%100!=0)  return(1);
           else if(year%400!=0)  return(0);
                 else return(1);
              break;
      default:return(0);
      }
    }
  void  main()
   {int month,year;
    printf(" 请输入月份：");
    scanf("%d",&month);
    switch(month)
    {
      case 1:
      case 3:
      case 5:
      case 7:
      case 8:
      case 10:
      case 12:printf("31 days\n");
      break;
      case 4:
      case 6:
      case 9:
      case 11:printf("30 days\n");
      break;
      case 2:printf("what is the year?");
       scanf("%d",&year);
       if(leapyear(year))
            printf("29   days\n");
       else printf("28 days\n");
            break;
     }
}
```

运行结果为：

请输入月份： 2↙

what is the year? 2000↙

29days

【例 7.18 】已知函数 fac 的原型为 long fac(int j)，其功能是利用静态变量实现 n！。要求编制该函数并用相应的主函数进行测试。

程序如下：

```
#include <stdio.h>
void main()
{     long fac(int j);
      int num,j;
      long ff;
      scanf("%d",&num);
      for(j=2;j<=num;j++)
      ff=fac(j);
      printf("num!=%ld\n",
ff);
   }
```

```
long fac(int j)
{
   static long ff=1;
   ff*= j;
   return ff;
 }
```

7.8 上机实训

❖ **实训目的与要求：**

- 熟悉 C 语言函数的定义、函数的声明以及函数的调用方法；
- 了解主调函数与被调函数之间的参数数递方式；
- 掌握变量的作用域和变量存储属性在程序中的应用；
- 能用自定函数方式编写一般应用程序。

❖ **实训任务：**

1．　写一个函数，使给定的一个二维数组（ 3*4 ）进行转置，即行列互换，并输出换置前后的结果。

2．　编写一个求素数的函数，然后用主函数调用该函数来求 100 至 500 之间的所有素数，并统计素数的个数。

3．输入若干个以回车键结束的字符串，然后将它们按照相反顺序输出。用递归函数实现。

- **实训报告：**
  1．　提交源程序文件、目标文件和可执行文件。
  2．　提交书面实训报告：报告包括原题，流程图，源程序清单及实验收获（如上机调试过程中遇到什么问题和解决方法）和总结等。