第5章 函数

●要求：

1) 掌握函数的定义、声明与调用，熟悉递归函数的定义与使用；

2) 掌握"函数与数组"的相关操作。

3) 熟悉存储类型、生存期和作用域；

4) 熟悉编译预处理。

1

5.1 函数的基本概念

函数是一个逻辑上独立的完成指定功能的代码段。对函数使用者来说，常把函数看作一个"黑盒"，只知道要传送给函数加工的内容，以及函数执行后能得到什么结果。在程序语言中函数是一个在逻辑上独立的程序单位，它可以定义自己的局部对象，不受主函数或其它函数对程序对象命名的影响。函数可带有参数，使函数执行时，操作对象、求值方式等可以随不同调用的需要而改变。在程序设计中，函数常被抽象为一个操作模式，是对语言提供的语句的扩充。函数也为程序的层次构造提供有力支持，使设计新程序时，能在已有函数的基础上构造功能更强的函数和程序。

2

在设计程序时，通常采用的是逐步分解、分而治之的方法，也就是把一个大问题分解成若干个比较容易求解的小问题，然后分别求解。根据该想法，程序员在设计一个复杂的应用程序时，往往也是把整个程序划分为若干功能较为单一的子问题，并把完成子问题的代码段编写成函数。这样，凡程序中需要完成函数功能的地方，就可以调用函数来实现。

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。

3

利用函数，不仅可以实现程序的模块化，程序设计变得简单和直观，提高了程序的易读性和可维护性，使程序的层次结构清晰，便于程序的编写、阅读、调试，可以把程序中普通用到的一些计算或操作编成通用的函数，以供随时调用，这样可以大大减轻程序员的代码工作量。

函数是C语言的基本构件，是所有程序活动的舞台。

小结：

函数的特征：

外部：可以作为"黑盒子"处理，根据不同的参数调用得到不同的结果。

内部：可以定义局部对象，不受主函数及其他函数影响。

4

【例5.1】已知圆柱体的半径和高，求圆柱体体积的程序。

```
#include <stdio.h>
int main()
{
   double  PI = 3.1415926, radius, height, vol;
   printf("输入圆柱体的半径和高\n");
   scanf("%lf%lf", &radius, &height);
   vol = PI * radius * radius * height;
   printf("圆柱体的体积是%f\n", vol);
   return 0;
}
```

5

如果该段程序在不同的地点要多次执行该项操作，可将计算圆柱体的体积的这部分代码抽象出来，定义成函数并给它一个名字，从而达到简化程序的目的。例如：

返回值的类型　函数名　　形参

```
double   volume(double radius, double height)
{   double  PI = 3.1415926, vol;
    vol = PI * radius * radius * height;
    return vol;
}
```

```
#include <stdio.h>
void main(){ double PI = 3.1415926, x, y, v;
  printf("输入圆柱体的半径和高\n");
  scanf("%lf%lf", &x, &y);
  v= volume(x,y);//x,y称为实参
  printf("圆柱体的体积是%f\n", v);
}
```

6

上述代码的作用是，如果程序的某个地方已知圆柱体的半径x和高度y，需要计算圆柱体的体积v。就可用以下函数调用代码实现计算圆柱体的体积：

```
v = volume(x, y);
```


5.2 库函数的使用方法

C语言提供了极为丰富的库函数，这些库函数又可从功能角度作以下分类。

- ◆输入输出函数：用于完成输入输出功能。
- ◆字符串函数：用于字符串操作和处理。
- ◆内存管理函数：用于内存管理。
- ◆数学函数：用于数学函数计算。

调用库函数必须在文件头部写上"包含"相应头文件的预处理命令。

- ◆#include <stdio.h> //输入输出库函数
- ◆#include <math.h> //数学库函数

以下是常用的头文件：

- ●stdio.h 输入输出库函数
- ●math.h、stdlib.h、float.h 数学库函数
- ●time.h 时间库函数
- ●ctype.h 字符分类和转换库函数
- ●string.h 内存缓冲区和字符串处理库函数
- ●malloc.h、stdlib.h 内存动态分配库函数

【例5.2】产生10个0～100之间的随机数。

```
#include <stdio.h>  /* 输入输出库函数的头文件 */
#include <time.h>   /* 时间库函数的头文件 */
#include <stdlib.h> /*数学或内存分配库函数的头文件 */
int main()
{  int k; long now;
   srand(time(&now));//返回当前日历时间
/* 用时间初始化随机数发生函数的初态，使初态总不相同 */
  for(k = 0; k < 10; k++) /*产生10个100以内的随机数 */
       printf("%d\n", rand()%100); /*调用随机函数 */
    return 0;
 }
/*clock():返回程序开始执行后占用的处理器时间 */
```


【程序说明】

rand()函数产生一个0~32767之间的随机数；

time()函数将从1970年1月1日00:00:00到当前时间所经过的秒数存储到实参指向的变量；

srand()函数用于重新设定rand()函数所使用的种子。

随机函数rand()生成的随机数的随机数种子由函数srand()设定。随机数的种子不同，由rand()函数产生的随机数序列也不相同。为了让程序每次运行产生的随机数不会相同，必须设置不同的随机数种子，用依赖于时间的值设定随机数种子，是最简单，也是最有效的方法。

从前面的学习中，得到C函数从不同的角度，有不同的区分：

从使用角度 → 系统函数，用户函数

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。在前面各章的例题中反复用到printf、scanf、strlen、strcat、pow(x,y)、fabs(s)等函数均属此类。

由用户按需要写的函数。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类型说明，然后才能使用。例如：前面的求体积中的函数：volume

从返回值 → 返回值函数，无返回值函数

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。如数学函数即属于此类函数。由用户定义的这种要返回函数值的函数，必须在函数定义和函数说明中明确返回值的类型.

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返回为"空类型"，空类型的说明符为"void"。

13

```
double  volume(double radius, double height)
{  double  PI = 3.1415926, vol;
   vol = PI * radius * radius * height;
   return vol;  //有返回值
}
```

```
void main()
{     ....
      return; //无返回值
}
```

14

从形式上 → 有参函数，无参函数

无参函数：

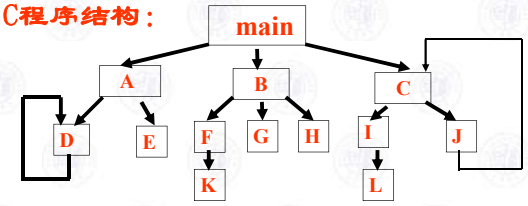
函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送。此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。 void main(){....}

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。pow(x, y)；

15

C程序结构：



- ☑ 程序中只能有一个主函数main()，不能被其它函数所调用；
- ☑ 主函数main()可以调用库函数或其他函数.或main()；
- ☑ 除主函数main()外，其他函数之间可以相互调用；
- ☑ 在一个程序中，通过调用关系将各函数联系在一起，程序总是从main()函数开始执行（于它的位置无关），调用所需要的函数，完成所调用函数的功能，返回到main()函数继续执行，最后在main()函数中结束。

16

主函数以main()函数作为程序的入口，从它开始执行。

在C语言中，函数不能嵌套定义，一个函数不从属于另一个函数，但函数可以相互调用。

一个C程序可由若干个源程序文件组成，每个源程序文件由一系列数据类型定义和说明、变量定义和说明、函数定义和说明等C代码组成。C程序的一个源程序文件对应通常所说的程序"模块"。一个源程序文件也是可独立编译的单位，C程序可以按函数分别编写，按源程序文件分别编译。

17

5.3 函数定义

将完成一定功能的算法编写成函数，称为函数定义。

函数定义的一般形式:

函数类型说明 函数名（形参说明表）

{

说明部分;

执行部分;

}

函数体

说明部分是对函数中要用到的变量、要调用到的函数以及要引用的外部变量进行说明。

执行部分是用来实现函数的功能，即语句执行部分。

18

返回值类型　函数名　形式参数

```
int max (int x, int y)
{  int z;
   z = x > y ? x : y;
   return(z);
}
```

函数体　返回值

函数的主要部件
- 函数名
- 形式参数
- 函数体
- 返回值

除主函数main()外，其他函数名是由用户定义的标识符 。
可将函数说明为返回任何一种合法的C语言数据类型。类型说明符告诉编译程序它返回什么类型的数据。当函数执行不返回值时，习惯用void来标记；返回语句return有两个重要用途。
第一，它使得内含它的那个函数立即退出，也就是使程序返回到调用语句处继续进行。第二，它可以用来回送一个数值.
注意：函数不能作为赋值对象，下列语句是错误的：
max(x ,y) =100;

【例5.4】求两个正整数最大公因子的函数gcd()

【解题思路】

两个正整数a和b的最大公因子有性质：

$$gcd(a, b) = \begin{cases} gcd(a-b, b), & \text{如果}a > b; \\ gcd(a, b-a), & \text{如果}a < b; \\ a, & \text{如果}a = b。 \end{cases}$$

求两个正整数最大公因子的函数可描述如下：

```
int gcd(int a, int b)
{   while( a != b)//直至a和b相等结束
    if(a > b) a -= b;
     else b -= a;
    return a;
}
```


采用辗转相除法求最大公因子，有以下算法：

A. [求余数]求a除b的余数r；

B. [判结束]如r等于0，b为最大公因子，算法结束；否则执行步骤C；

C. [替换]用b置a，r置b，并回到步骤A。

```
int gcd(int a, int b)
{  int r;
   while(r = a % b){/*求余数，判是否结束*/
     a = b;  b = r; /* 完成：替换*/
   }
   return b;
}
```

该函数的类型说明gcd ( )函数返回整数类型的数据。

(1) 函数无返回值

如果函数没有返回值，则一般在定义函数时把“函数类型说明符”说明为void。

【例5.5】输出换行符的函数printnl。

```
void  printnl( void)
{
      printf("\n");
}
```

【程序说明】

函数printnl()不返回结果，也没有形参。

(2) 函数有1个返回值

这时在被调用函数的函数体中有return语句。此时函数类型的定义应根据return语句后的表达式的数据类型来定义。例如：

```
int max(int a,int b)
{
    return a>b?a:b;
}
```

注：

♦ 当函数类型与return语句后的表达式的数据类型不一致时，函数类型决定return语句后的表达式的数据类型，系统自动将表达式的数据类型转换成函数定义的数据类型.

在函数定义过程中，最好定义两者一致，以免结果出错；

```
int max(float a,float b)
{
    return a>b?a:b;
}
```

```
void main( )
{
    printf("%d",max(12.3,20.6));
}
```

输出结果：20

♦当缺省函数类型定义时，系统默认函数类型为int或char，同时也说明当函数类型为int或char时，可缺省函数类型说明。

```
max(int a, int b)
{
    return a>b?a:b;
}
```

(2) 函数有多个返回值

这时在被调函数的函数体中一般无return语句。多个值的返回是通过全局变量、数组或指针作参数来实现的，这将在以后的章节中进行介绍。

● 形参表及形参说明

在函数定义中写在圆括号中的参数称为形式参数。不带参数的函数其形参表是空的。一般无参函数在圆括号中可以写上"void"以取代空的形参表。例如：

```
void PRINT(void)
{
    printf("This is a example\n");
}
```

有参函数的形参表由一个或多个形参组成，各参数间用逗号分隔。如果有形参，则必须定义形参的数据类型。

```
函数类型 函数名(形参表及类型)
{
    函数体;
}
```


5.4 函数的调用

函数被定义以后，凡要实现函数功能的地方，就可简单地通过函数调用来完成。

1.函数调用的一般形式:

函数名(实际参数表)

实在参数：实参按顺序与函数定义中的形参一一对应，且类型一致。实际参数表中的参数可以是常数，变量或其它构造类型数据及表达式。各实参之间用逗号分隔。

对无参函数调用时则无实际参数表。

若调用无形参的函数，调用形式变为：

函数名()

说明：函数名之后的一对圆括号不能省略。

2. 函数调用有以下几种方式：

(1) 函数语句

```
printf("Hello,world!\n");
scanf ("%d",&a);
```

(2) 函数表达式

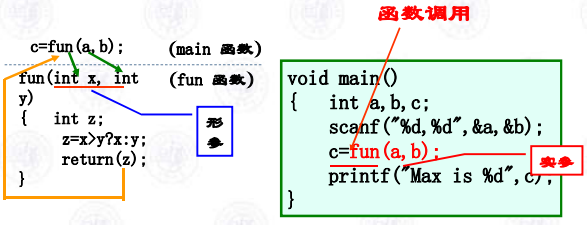
```
c=max(a,b);
```

(3)函数参数

```
minvalue=min(a, min(c, d));
//对返回值进一步计算
printf("%f\n",min(u-v,a+b));
/*直接输出函数的返回值*/
```


3. 函数调用的执行过程

举例：函数应用



♦函数调用的图示

设有main函数、fun函数，则它们的调用过程如图所示。

函数调用过程简述

① 首先保存断点有用信息，为被调函数的所有形式参数分配内存，再计算实际参数的值，一一对应地赋给相应的形式参数(对于无参函数，不做此工作)；

② 然后进入被调函数的函数体，为函数说明部分定义的变量分配存储空间，再依次执行函数体中的可执行语句；

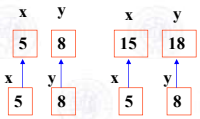
③ 当执行到"return"语句时，计算返回值(如果是无返回值的函数，不做这项工作)；释放本函数中定义的变量所占用的存储空间(对于static类型变量，其空间不释放)，返回主调函数继续执行。

4. 实参向形参传递数据

函数间通过参数传递数据，是通过调用函数中的实在参数（简称实参）向被调用函数中的形式参数（简称形参）传递进行的。

实参向形参传递数据的方式：是实参将值单向传递给形参，形参值的变化不影响实参值。

♦简单变量作函数的实参



【例5.8】值传递的示意程序。

```
#include <stdio.h>
void func(int x, int y)
{ x += 10;    y += 10;
  printf(" 在func函数中：x = %d,  y = %d\n", x,y);
}

int main()
{  int x = 5, y =8;
   printf(" 在主函数中x与y的初值是：x = %d,  y = %d\n",x,y);
   func(x, y);
   printf("调用func函数后返回到主函数时：x=%d,y=%d\n",x,y);
   return 0;
}
```


程序运行结果如下：
在主函数中,x与y的初值是,x=5,y=8
在fun函数中,x=15,y=18
调用fun函数后返回到主函数,x=5,y=8

【程序说明】

从上面的例子可以看到，实参变量和形参变量尽管名称是相同的,但在内存中所占用的存储空间是不相同的。因此，调用func()时，函数的形参接收了实参的值，并对形参的值进行了修改，但不会传回给实参变量。

♦一维数组元素作函数的实参

由于数组元素与相同类型的简单变量地位完全一样，因此，数组元素作函数参数也和简单变量一样，也是值的单向传递。
如前例:

```
int main()
{  int x[2] = {5, 8};
   printf(" 在主函数中x1与x2的初值是：x0= %d,  x1= %d\n",
          x[0],x[1]);
   func(x[0], x[1]);//传递数组元素，数组元素为实参
   printf("调用func函数后返回到主函数时：x0=%d,x1=%d\n",
          x[0],x[2]);
 return 0;
}
```

但当是数组名时,由于数组名代表的是数组首元素的地址，所以常用数组名实参对应数组形参,以达到值的修改。

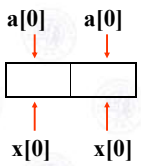
♦数组名作实、形参

1.在一维数组中，用"数组名［下标］"表示的数组元素相当于一个普通变量；而不带下标的数组名代表一批变量，也可以把它看成一个特殊的变量，因为它存放该数组的首地址（即数组第一个元素的地址）。

2.在函数中，直接用数组名作参数时，则传送的是地址值，即把实参数组的首地址传递给形参数组，而不是将全部数组元素都复制到函数中去。地址传递后，实参、形参数组共享相同的内存单元，也就是说形参数组和实参数组其实就是同一个数组。

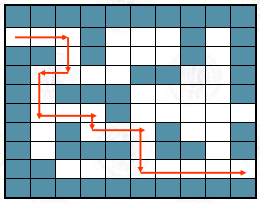

```
void swap( int x[])
{
        int j;
        j=x[0];
        x[0]=x[1];
        x[1]=j;
}

void main( )
{  int a[2]={5,10};
   swap(a);
   printf("a=%d,b=%d\n",a[0],a[1]);
}
```

在本例中，实参数组名为a，形参数组名为x。在主程序的函数调用（swap(a)）时，实参数组a[2]通过数组名a把首地址传给x后，则x和a共享实参数组所占用的内存空间，也就是说形参数组x和实参数组a其实就是同一个数组，只不过是有两个名字而已，如图所示。



迷宫问题

问题描述：如图所示的方块图表示迷宫，其中空白方块表示通道，灰色方块表示墙，在迷宫数组中分别用0和1表示。现要在迷宫中找一条从入口到出口的可通路径。



1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0	0	1
1	0	1	1	1	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	1
1	0	1	1	1	0	1	1	0	1
1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

算法分析：从入口处开始向前探路（探的方向有右、下、左和上，分别是(x, y+1)，(x+1, y)，(x, y−1)，(x−1, y)），当沿着某个方向往前探一步时，若通可则继续往前探，若不通则换个方向后再探。若四个方向上均不通（四个方向上的相邻方块要么是墙块，要么是已探过的通道块），则按原路回退一步后换个方向再探。在探路的过程中，用一个二维数组记录迷宫的可通路径。

37

算法说明：

算法中主要用到2个数组：

1）maze[N1][N2] ——迷宫数组。

0代表通（通道），1代表不通（墙）

2）stack[N1*N2][2] ——可通路径数组，记录走迷宫的路径，第1个下标表示是当前正在试探的可通路径上的第几步，第2个下标表示该步所在位置的坐标是行坐标x还是列坐标y。

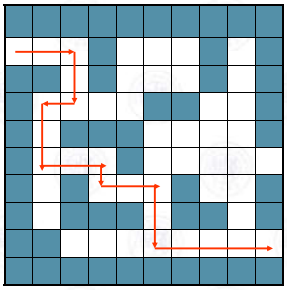
38

```
int getpath(int maze[N1][N2])  /*在迷宫中探路径的函数*/
{     int stack[N1*N2][2];
      int i,x=1,y=0,ok,top=0;
      /*可通路径上第一个点（入口点）的x和y坐标进入路径数组stack*/
      stack[top][0]=x;stack[top][1]=y;
   /*循环探路。top指向路径上的最新点，若top加1，则向前探；若top减1，则回退一步*/
      while(1)
      { ok=0;      /*ok标志表示能否往前走一步*/
        if (maze[x][y+1]==0) {y=y+1;ok=1;}       /*往右试探*/
        else if (maze[x+1][y]==0) {x=x+1;ok=1;}   /*往下试探*/
        else if (maze[x][y-1]==0) {y=y-1;ok=1;}   /*往左试探*/
        else if (maze[x-1][y]==0) {x=x-1;ok=1;}   /*往上试探*/
        if(!ok)  /*若4个方向都走不通，则往回退一个位置，top减1*/
        {    top--;
             if(top==0)   /*若top减1后为0，表示栈空，即迷宫无出路*/
             {   printf("迷宫中找不到可通路径!\n");return 0;}
                 /*下面2个语句是出栈*/
             x=stack[top][0];  /*回退后，路径上最后一个点的X坐标存入x*/
             y=stack[top][1];  /*回退后，路径上最后一个点的Y坐标存入y*/
        }
```

39

```
else /*若走得通，则将该点作上标记2，并该点坐标存入路径数组stack*/
{     maze[x][y]=2;      /*将已走过的点标为2*/
      top++;/*下面2个语句是将走通坐标保留*/
      stack[top][0]=x;/*新进入路径的点的X坐标进入路径数组stack*/
      stack[top][1]=y;/*新进入路径的点的Y坐标进入路径数组stack*/
      if (x==N1-2 && y==N2-1)/*到达出口，则输出路径*/
     {   printf("求得的迷宫路径为:\n");
         for (i=0;i<=top;i++)
         {   printf("(%d,%d)%s",stack[i][0],stack[i][1],i<top?"-->":"\n");
             if ( (i+1)%5==0) printf("\n");
         }
        return 1;
     }
  }
 }//对应while
}
```

40



1	1	1	1	1	1	1	1	1	1
0	2	2	1	2	2	2	1	0	1
1	1	2	1	2	2	2	1	0	1
1	2	2	2	2	1	1	0	0	1
1	2	1	1	1	0	0	0	0	1
1	2	2	2	1	0	0	0	0	0
1	0	1	2	2	2	1	0	0	1
1	0	1	1	1	2	1	1	0	1
1	0	0	0	0	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1

分析在探路的过程中数组maze和stack中数据的变化过程。按照程序探路方向的判断次序，一口气走了12步（红色的）（top=12），即到了x=1, y=4, 遇阻，此时要回退(top—，蓝色的)，直到top=4, x=3, y=2，它的左面x=3, y=1的位置是0,所以又可以继续探路了。程序关键是当四个方向都行不通的时候，要按原路回退，直到遇到新的路径，再继续进行。若回退到top=0，则表明迷宫中找不到可通路径，程序退出。

41

stack[N1*N2][2]的变化过程

top	x	y
0	1	0
1	1	1
2	1	2
3	2	2
4	3	2
5	3(3)	3(1)
6	3(4)	4(1)
7	2(3)	4(1)
8	2(5)	5(1)
9	2(5)	6(2)
10	1(5)	6(3)

top	x	y
11	1(6)	5(3)
12	1(6)	4(5)
13	7	5
14	8	5
15	8	6
16	8	7
17	8	8
18	8	9

. . . . . .

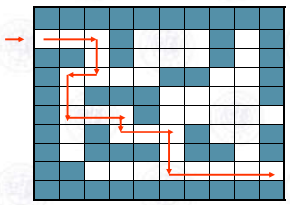
top	x	y

从表中得知，在top=12遇阻，然后回退，直到top=4，又找到了可以通的路径，蓝色的为回退路径，凡加括号的x, y的值为回退后的走的另一条路径。

42

此例，第0步（入口处）的x坐标为1，即stack[0][0]=1；第0步（入口处）的y坐标为0，即stack[0][1]=0。同理，第1步的x坐标和y坐标分别表示为stack[1][0]和stack[1][1]。依此类推，第n步的x坐标和y坐标分别表示为stack[n][0]和stack[n][1]。若记入口处坐标为（1,0），则本例执行后输出的可通路径上各点的坐标依次为：

→（1, 0）→（1,1）→（1, 2）→（2, 2）→（3, 2）→（3, 1）→（4, 1）→（5, 1）→（5, 2）→（5, 3）→（6, 3）→（6, 4）→（6, 5）→（7, 5）→（8, 5）→（8, 6）→（8, 7）→（8, 8）→（8,9）。



1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	0	1	0	1
1	1	0	1	0	0	0	1	0	1
1	0	0	0	0	1	1	0	0	1
1	0	1	1	1	0	0	0	0	1
1	0	0	0	1	0	0	0	0	0
1	0	1	0	0	0	1	0	0	1
1	0	1	1	1	0	1	1	0	1
1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

重点小结：

本节在分析函数调用过程的参数传递问题时，特别强调要搞清楚实参向形参传递的是"值"还是"地址"？严格说来，"地址"也是值（地址值），但它们有一个显著的区别：

① "值传递"在实参将"值"传递给形参后，对形参的修改不会影响到对应的实参。因为只有在被调用时才由系统给形参分配存储单元，在调用结束后，形参所占用的存储单元被释放；这可以理解为实参和形参各自占用不同的存储空间，实参在将"值"传递给形参后，二者就脱离关系了；

② "地址传递"在实参将"地址"值传递给形参后，形参就和实参共享同一地址单元，而不另外分配存储空间。这可以理解为形参名和实参名只是同一存储单元的两个不同引用名而已，因而对形参的修改就相当于是对实参的修改。

另外也请注意：

1. 实参的个数与类型应与形参一致，否则将会出现编译错误；
2. C编译系统对形参数组大小不作检查，因此形参数组可以不指定大小，在数组名后跟一对空的方括号即可，而其大小由相应的实参数组决定。

函数的作用域规则

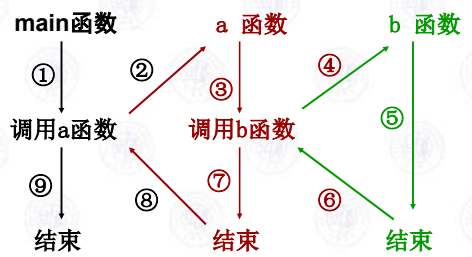
C语言中的每一个函数都是一个独立的代码块。一个函数的代码块是隐藏于函数内部的，不能被任何其它函数中的任何语句（除调用它的语句之外）所访问（例如，用goto语句跳转到另一个函数内部是不可能的）。构成一个函数体的代码对程序的其它部分来说是隐蔽的，它既不能影响程序其它部分，也不受其它部分的影响。换言之，由于两个函数有不同的作用域，定义在一个函数内部的代码数据无法与定义在另一个函数内部的代码和数据相互作用。

C语言中所有的函数都处于同一作用域级别上。这就是说，把一个函数定义于另一个函数内部是不可能的。

5.函数的嵌套调用

前面介绍C语言的函数定义都是平行、独立的。即在一个函数内不能包含另一个函数。C语言中不允许嵌套的函数定义，但可嵌套调用。关系可表示如图。

函数嵌套调用示图



特点是：

后调用的，先返回。

【例5.10】函数嵌套调用的示意程序。

```
#include <stdio.h>
#include <math.h>
int istri(float a, float b, float c) /* 判是否可构成三角形函数 */
{ if ( a+b<=c || a+c<=b || b+c<=a ) return 0;/* 不能构成三角形 */
  if ( a <= 0 || b <= 0 || c <= 0 ) return 0;/* 不能构成三角形 */
  return 1;   /* 能构成三角形 */
}

float triangle(float a, float b, float c)  /* 求三角形面积函数 */
{ float s, area;
/* 调用istri()函数，查看是否能构成三角形 */
  if (istri(a, b, c) == 0) return 0; /* 返回0不能构成三角形 */
  s = (a+b+c)/2;
  area = sqrt(s*(s-a)*(s-b)*(s-c));
  return area;/* 返回已计算的三角形面积 */
}
```

```
int main()    /* 主函数 */
{ float a, b, c, area;
  do
  { printf("请输入三角形三条边a, b, c: ");
    scanf("%f%f%f", &a, &b, &c);
    area = triangle(a, b, c);/* 调用triangle()函数 */
    if(area == 0)printf("输入数据错，不能构成三角形! \n");
  } while(area == 0);  /* 不能构成三角形，重新输入数据 */
  printf("三角形的面积为: %f\n", area);
  return 0;
}
```

请输入三角形三条边a, b, c：1,2,3
输入数据错，不能构成三角形!
请输入三角形三条边a, b, c：3,4,5
三角形的面积为：6.000000

● 函数调用的规则说明

- 被调用的函数必须是已经存在的函数(即是库函数或用户自己定义的函数)；
- 如果调用库函数，需要在程序的开头包含相应的头文件，如使用数学库中的函数，就用#include<math.h>；
- 函数的声明：

5.5 函数说明

● 函数声明的含义

函数已定义，如果要调用，一般应在主调函数中对被调函数进行声明，即向编译系统声明将要调用此函数，并将有关信息（如被调用函数名、函数类型、形参的个数及类型等）通知编译系统。

编译程序根据这些信息检查调用的正确性。对不正确的调用给出错误信息，对正确的调用编译出实现的机器代码。

调用函数与被调用函数之间在程序正文中可能会存在以下几种情况。

（1）调用同一程序文件中前面已定义的函数。
（2）调用处于同一程序文件后面定义的函数。
（3）调用别的程序文件中定义的函数。

对于第一种情况，在函数调用处，被调用函数的详细信息已被编译程序所接受。
对于后两种情况，这时因被调用函数的信息还未被编译程序所接受，不能检查函数调用的正确性，所以在调用之前需对被调用函数有关调用的一些信息作出说明。

函数与调用有关的信息包括：
函数的返回值类型、函数名和函数有关形参的个数及其类型等。
只给出函数的调用信息称作函数说明。

● 函数声明的一般形式

(1)函数类型 函数名(形参类型1 参数1，形参类型2 参数2， ...)；

如：double power(double a , int b)；

(2)函数类型 函数名(形参类型1，形参类型2，...)；

如：double power(double , int)；

因编译系统不检查参数名，故格式(1)(2)效果一样

其中：类型：该函数返回值的类型。
●形参类型表：顺序给出各形参的类型，如果函数没有形参，形参类型表可以为空。
为了强调函数没有形参，空形参类型表可以写成void。

例求两个数之和

```
#include <stdio.h>
 void main( )
{
   float fun(float x,float y);
   printf("Sum=%f\n",fun(2,5));
}
 float fun(float x,float y)
{
   return x+y;
}
```

Sum=7.000000

```
#include <stdio.h>
 void main( )
 {
   float fun(float,float);
   printf("Sum=%f\n",fun(2,5));
 }
   float fun(float x,float y)
   {
      return x+y;
   }
```

Sum=7.000000

例　求两个数之和。

```
#include <stdio.h>
void main( )
{
    float fun( );  //形参类型没有说明
    printf("Sum=%f\n",fun(2,5));
}
 float fun(float x, float y)
{
     return x+y;
}
```


对于用户自定义函数，若被调用函数定义在主调函数之前，可缺省声明。

例：　求两个数之和。

```
#include <stdio.h>
float fun(float x,float y)
{
     return x+y;
}
void main( )
{
     printf("Sum=%f\n",fun(2,5));
}
```


- 关于函数声明的小结

在今后的编程实践中，养成如下习惯就不会出现意想不到的错误。

- 将自定义函数在主调函数之前进行定义，则可缺省函数声明。
- 若自定义函数在主调函数之后进行定义，则严格按照两种声明格式在调用前进行函数声明。

5.6 递归函数基础

递归函数概念

C 语言允许函数直接或间接地调用它自身，这种调用形式称为递归调用。含有递归调用的函数称为递归函数,也可称为“循环定义”。若函数在本函数体内直接调用本函数，称为直接递归
，如图a 所示。若某函数调用其它函数，而其它函数又调用了本函数，这一过程成为间接递归。如图b 所示，在a 函数中调用b 函数，在b 函数中又调用a 函数。

递归函数示意图

直接递归a　　间接递归b

【例5.12】用递归实现阶乘计算函数

用循环计算阶乘 n! 的思想是：

从1开始，乘2、再乘3、…、一直乘到n为止。

用递归计算阶乘 n! 的思想是（设 n = 4）：

4!等于4×3! ，而3!等于3×2! ，…，1!=1

n!=n*(n-1)!,(n-1)!=(n-1)*(n-2)!...,

阶乘函数的递归公式及函数如下：

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

用自身的结构来描述自身就称为"递归"。

用递归实现阶乘

```
#include <stdio.h>
int fac(int n)
{
     if(n<=1)return 1;
  return n*fac(n-1);
}

void main( )
{
  int n;
  scanf("%d",&n);
  printf("%d!=%d\n",n,fac(n));
}
```

用循环实现阶乘

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

非递归函数fac( )的执行应该是易于理解的。它应用一个从1开始到指定数值结束的循环。在循环中，用“变化”的乘积依次去乘每个数。

若程序中输入的n值为3，则在main( )函数中调用了fac(3)，其调用过程如下：

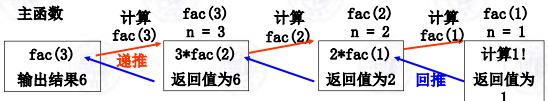


图 求3！递归调用的过程

递归执行比循环稍复杂。当用参数1调用fac( )时，函数返回1；除此之外的其它值调用将返回fac(n-1)*n这个乘积。

为了求出这个表达式的值，用(n-1)调用fac( )一直到n等于1，调用开始返回。计算2的阶乘时对fac( )的首次调用引起了以参数1对fac( )的第二次调用。这次调用返回1，然后被2乘，则fac(2) 的答案是2.

递归计算n!有两个重要的求解过程：

一是"递推"过程

为求n!的解，去求(n-1)!的解，求完(n-1)!的解又继续去求(n-2)!的解，依此类推，最终要求1!的解。将求大规模问题解，演变为求规模略小问题解的"递推"过程。

二是"回推"过程

有了1!的解后，逐步得到2!的解、3!的解、直到n! 的解。从上面所述的两个过程可以看出，在递归调用过程中如果没有递归终止条件1!=1，则回推过程始终不会结束，即无穷回推，这样就不能得到结果。

递归函数的主要优点是可以把算法写的比使用非递归函数时更清晰更简洁。编写递归函数的步骤：

1：描述递归关系，例如计算阶乘： n*fac(n-1)

2：确定递归边界(递归出口)，必须在函数的某些地方使用if语句，强迫函数在未执行递归调用前返回。如果不这样做，在调用函数后，它永远不会返回。在递归函数中不使用if语句，是一个很常见的错误。

特别指出，虽然递归是直接或间接的调用自身，但和调用其它函数一样需要重新开辟内存空间，因此调用自身和调用其它函数的调用过程无区别。编写递归程序比较简单，关键是确定递归公式。由于递归程序需要使用大量的存储空间，因此它的执行效率较低。

- 定义递归函数的一般形式

形式1：

```
返值type  fun(参数说明表)
   {
      if(递归终止条件)
         返回值p = 递归终止值；/*递归终止*/
      else
         返回值p = 递归fun调用(参数) ；/*递归调用*/
      return   p;
   }
```


形式2：

```
返值类型  fun(参数说明表)
{
   if(递归终止条件)
      return    递归终止值；//递归终止
   else
      return    递归调用fun(参数)的表达式；//递归调用
}
```

```
int  fac(int  n )
{  if( n == 1 )        /*递归终止条件*/
      return  1;    /*递归终止值*/
   else
   return( n * fac(n-1)); /*递归调用*/
}
```

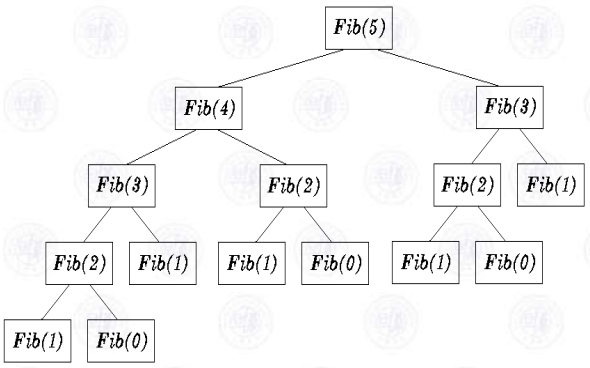

【例5.13】计算斐波那契数列的第n项

斐波那契数列函数*Fib*(*n*)的定义

$$Fib(n)=\begin{cases} n, & n=0,1 \\ Fib(n-1)+Fib(n-2), & n>1 \end{cases}$$

求解斐波那契数列的递归算法

```
long  Fib( long   n )
{
      if(n==0 || n==1)
         return  n;
      else
         return   Fib(n-1) + Fib(n-2);
}
```

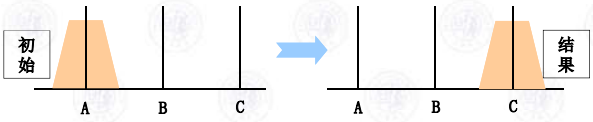


斐波那契数列的递归调用树

67

【例5.14】求解的Hanoi塔问题

汉诺塔（Tower of Hanoi）游戏据说来源于印度神庙。游戏的装置如图所示（图上以3个金片例），底座上有三根金的针，第一根针上放着从大到小64个金片。游戏的目标是把所有金片从第一根针移到第三根针上，第二根针作为中间过渡。每次只能移动一个金片，并且大的金片不能压在小的金片上面。该游戏的结束就标志着“世界末日”的到来。



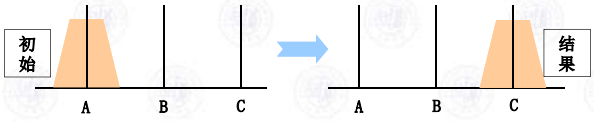
问题：

有三根针（设为A、B、C），在A针上有n张大小均不相同的盘子，大的在下，小的在上。

68

题解： 游戏中金片移动是一个很繁琐的过程。通过计算，对于64个金片至少需要移动 $2^{64} - 1 = 1.8\times1019$ 次 。

不妨用A表示被移动金片所在的针（源），C表示目的针，B表示过渡针。对于把n（n>1）个金片从第一根针A上移到第三根针C的问题可以分解成如下步骤：



圆盘移动规则

1）每次只能移动一个圆盘；

2）圆盘可以插在A, B和C中的任一针上；

3）任何时刻都不能将一个较大的圆盘压在较小的圆盘上。

69

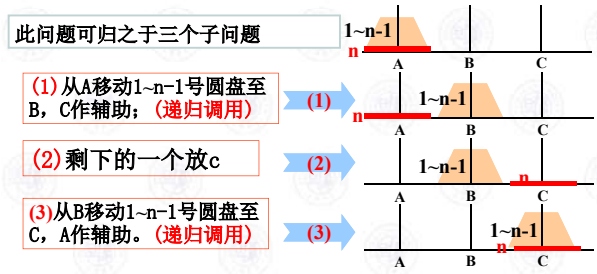
汉诺塔(Tower of Hanoi)问题的递归解法

此问题可归之于三个子问题

(1) 从A移动1~n-1号圆盘至B，C作辅助；（递归调用）

(2) 剩下的一个放c

(3) 从B移动1~n-1号圆盘至C，A作辅助。（递归调用）



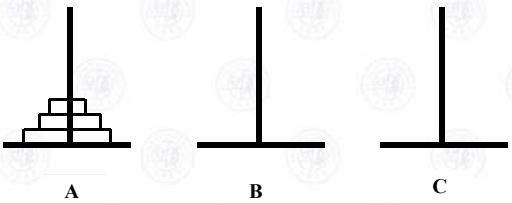
其中:1和3 是递归,直到一个盘子.

搬动次数2^n-1

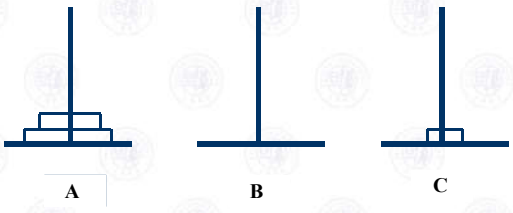
70

汉诺塔(Tower of Hanoi)问题

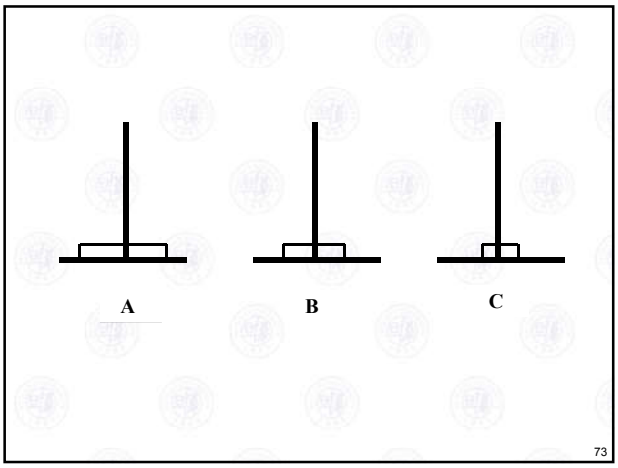
例如：观察3个盘子移动的过程：

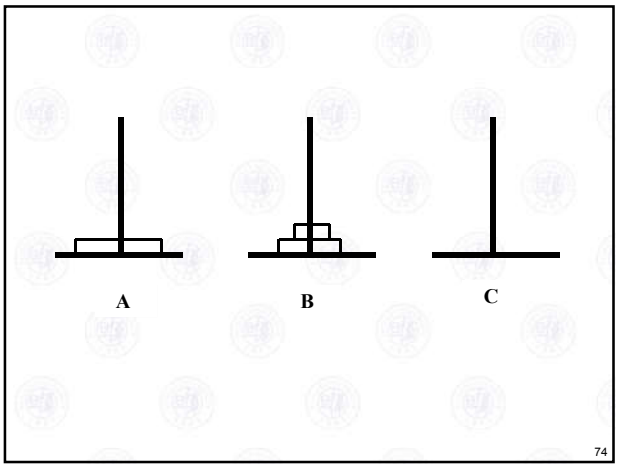


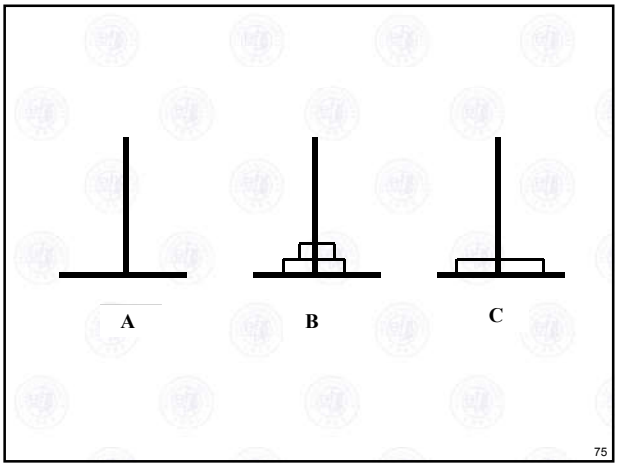
71

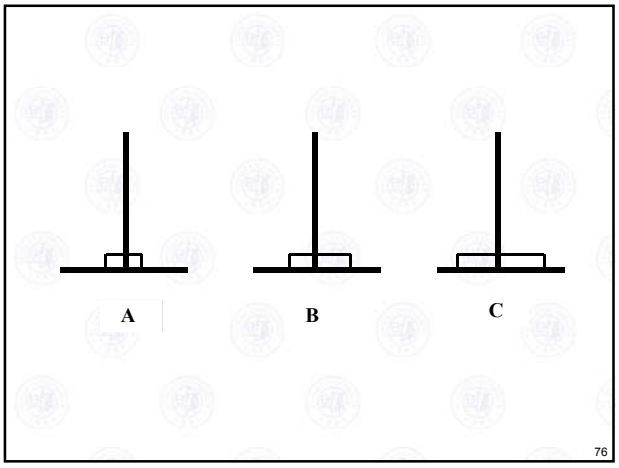


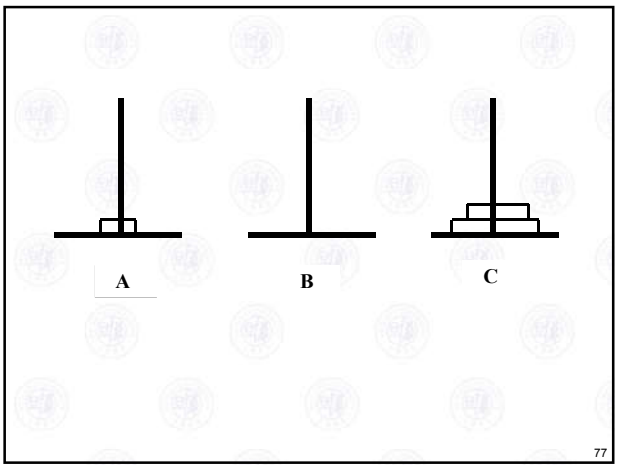
72

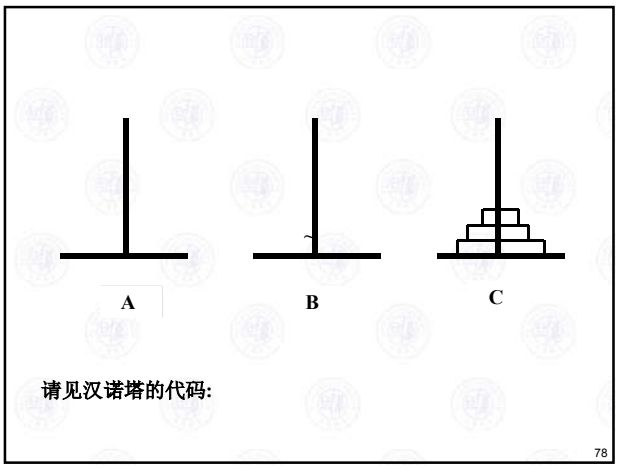

A
B
C
73


A
B
C
74


A
B
C
75


A
B
C
76


A
B
C
77


A
B
C
请见汉诺塔的代码:
78

```
#include <stdio.h>
 void move(int m,char from,char to)
 {   printf("%d号盘子从%c --->%c\n",m,from,to);
              //只有一个盘子从a搬到c
 }
 void hanoi(int n,char A,char C,char B)//递归函数
 { if( n==1 ) move(n,A,C);
   else
   { hanoi(n-1,A,B,C); //将n-1个盘子从A搬到B,借助C
     move(n,A,C); //将A上最后一个盘子搬到C
     hanoi(n-1,B,C,A ); //将n-1个盘子从B搬到C,借助A
   }
 }

void main()
{
    hanoi(3,'A','C', 'B');
}
```

A, C, B

```
3:A,C,B
2:A,B,C
1:A,C,B
1 号盘子 A--->C
2 号盘子 A--->B
1:C,B,A
1 号盘子 C--->B
3 号盘子 A--->C
2:B,C,A
1:B,A,C
1 号盘子 B--->A
2 号盘子 B--->C
1:A,C,B
1 号盘子 A--->C
```


递归在算法上简单而自然，递归过程结构清晰，源程序代码紧凑，因而递归调用在完成诸如阶乘运算、级数运算以及对递归的数据结构进行处理等方面特别有效。

递归与递推的区别

- 递推方法
  - 主要采用循环技术；
  - 逐步执行；
  - 当前值的求得总建立在前面求解的基础上；
  - 占用存储空间少，执行速度快。
- 递归方法
  - 描述与原始问题（递归公式）比较接近；
  - 书写简洁、易读易写；
  - 易于分析算法的复杂性和证明算法的正确性；
  - 在问题转化时，需要花时间和存储空间将有关的“现场信息”保存起来；当达到中止条件时，系统又需要花时间将有关的“现场信息”恢复以便处理未曾处理完的函数调用 。

5.7 函数程序设计实例

【例5.14】求一整数的十进位数。

【解题思路】

求n是几位十进位数，只要反复将n除以10，直至n等于0。循环除10的次数，就能推算出n的十进位的位数。函数开始时预置计数器c为0，循环的工作部分是让c增1和n除以10，循环直至n除以10后为0结束。

```
int digits(int n)
{ int c=0;
  do {
     c++;  n/=10;
  } while(n);
  return c;
}
```

```
#include <stdio.h>
int main()//主函数
{ int n;
  printf("请输入一个整数：");
  scanf("%d", &n);
  printf("%d有%d位十进位数字.\n",
          n, digits(n));
  return 0;
}
```


【例5.15】判断一个十进制整数是否是回文数。

【解题思路】

所谓"回文数"是指左右对称的数字序列，即自左向右读和自右向左读是相同的数。例如，232、353、12321等都是回文数。

方法一：将整数n的各位数字拆开按顺序存入一数组中，然后依次将其首末对应位置中的数字两两比较，若对应位数字都相同，则n是回文数；否则，n不是回文数。


```
#include <stdio.h>  //方法1
int circle(int n)
{ int t[12],k = 0, j;
  do { t[k++] = n%10;
  // 取余数,依次存入t数组中
       n /= 10;              /* 取商*/
  } while(n);
  for(j = 0, k--; j < k; j++, k--)
    if(t[j] != t[k]) return 0;/* 不是回文数 */
  return 1;                  /* 是回文数 */
}
```

```
void  main()
{ int c=123321;
  if(circle(c))
     printf("%d是回文数 ", c);
  else printf("%d不是回文数 ", c);
}
```

```
//方法2，习题p67_23已讲解
 int circle(int n)
{      int s=0,m=n;
       while(m)
       {    s=s*10+m%10;   //颠倒乘,组成新整数
            m/=10;          //取新的被除数
       } return s==n;
}
```


【例5.16】编写一个函数验证哥德巴赫猜想，任何一个不小于6的偶数均可以表示为两个素数之和。如，6 = 3 + 3，8 = 3 + 5，10 = 3 + 7，… 。程序要求输入一个偶数，输出6到该偶数范围内各个满足条件的组合。

【解题思路】

对于一个偶数n，分解为两个素数之和的一般形式为n=x+y。从为x查找最小的素数开始（在这里x从3开始，不能取2，否则y也为偶数了），再判断y是否为素数，如果y是素数，则找到了这两个素数。如果不是，重新为x找下一个素数，再判断y是否为素数。

```
#include <stdio.h>
int isPrime(int n) //判别n是否为素数的函数
{ int k;
  if (n == 1) return 0;   /* 1不是素数 */
  if (n == 2) return 1;   /* 2是素数    */
  if (n % 2 == 0) return 0;// 除2外，其他偶数不是素数
for(k = 3; k*k <= n; k += 2)
   if (n % k == 0) return 0; //n能整除k，则n不是素数
return 1; // 所有k都不能被n整除，则n为素数
}
```

85

```
int main()
{ int n, x, y, m, count=0;
  printf("请输入一个不小于6的偶数：");
  scanf("%d", &m);
  for( n = 6; n <= m; n += 2)
    for( x = 3; x <= n/2; x+=2)
      if(isPrime(x) && isPrime(y = n - x))
      {
         if(count++ % 4 == 0) printf("\n");
            printf("%d = %d + %d\t", n, x, y);
         break;       /* 跳出内循环 */
      }
  printf("\n");
  return 0;
}
```

86

【例5.17】递归计算x的y次方。

【解题思路】

在递归函数的“递推”过程中，将求x的y次方不断地分解为求x的y-1次方，最终求解x的0次方为1；在“回归”过程中，已知x的0次方解，不断地再乘以x，最终得到x的y次方的解。

```
#include <stdio.h>
int power(int x, int y) /* 计算x 的 y 次方的递归函数 */
{ if(y == 0) return 1;  /* 任何不等于0的数的0次方为1 */
  return x*power(x, y-1);
}
int main() /* 主函数 */
{ int x, y;
  printf("请输入 x(!=0) 和 y ： ");
  scanf("%d%d", &x, &y);
  printf("%d^%d = %d\n", x, y, power(x, y));
  return 0;
}
```

87

【例5.18】输入一个正整数，用递归实现将该整数倒序输出。

【解题思路】

先输出整数的个位，接着用递归倒序输出高位即可，见下面程序中的函数back()。

```
#include <stdio.h>
void back(int n)              /* 求倒序的递归函数 */
{   printf("%d", n%10);  /* 输出尾数 */
    if(n < 10) return;
    back(n/10);
}
int main()   /* 主函数 */
{ int x;
 do {printf("请输入要倒序的正整数："); scanf("%d",&x);
 } while( x <= 0);
 printf("倒序数："); back(x); printf("\n");
 return 0;
}
```

88