第4章 数组

●要求：

1）掌握一维、二维及多维数组的定义、初始化与引用；

2）掌握字符串数组与字符串操作；

4.1 数组的基本概念

C语言提供了多种数据类型，除了前面介绍的基型、实型和字符型等基本数据类型外，还有一些扩展的数据类型，如**数组**、**指针**、**结构和联合等**。由于它们是**由基本数据类型按一定规则组成的**，所以被称之为**复合数据类型或构造数据类型**。

本章首先介绍一种最常用的构造数据类型—数组。

什么是数组?

数组是有序数据的集合。数组中的每一个元素都属于同一个数据类型。用一个统一的**数组名**和**下标**来唯一地确定数组中的**元素**。

最低地址对应于数组的第一个元素，最高地址对应于最后一个元素。

- 按照数组元素的类型可把数组分为**基型数组**、**实型数组**、**字符型数组**和**指针型数组**等。
- 按照下标的个数又可以把数组分为**一维数组**、**二维数组**和**多维数组**。

我们先学习最简单也是最常用的——**一维数组**

4.2 一维数组

在C语言中，与变量的定义一样，数组也遵循"先定义后使用"的原则。当定义数组时，要传递给编译器两方面的信息：

① 数组共有多少个元素？

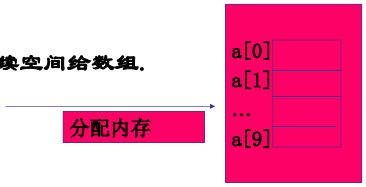
② 每个元素占多少个字节？

根据以上信息，编译器决定分配多大的存储空间给该数组使用。

例：int a[10];

这里a是数组的名称，方括号中的10表明数组一共有10个元素，下标应该从0开始到9结束；类型名int限定数组a的每个元素中只能存放基型数。根据这一定义，系统将为数组a开辟能容纳10个基型数的连续存储单元。

在内存分配若干连续空间给数组.



用printf我们可以查看数组地址情况

```
# include <stdio.h>
void main( )
{ int  a[10],i;
  for(i=0;i<10;i++)
     printf("&a[%d]=%x\n",i,&a[i]);
}
```

▪运行结果

```
&a[0]=12ff58
&a[1]=12ff5c
&a[2]=12ff60
&a[3]=12ff64
&a[4]=12ff68
&a[5]=12ff6c
&a[6]=12ff70
&a[7]=12ff74
&a[8]=12ff78
&a[9]=12ff7c
```

●一维数组的定义格式为：

类型说明符　数组名[常量表达式]；

说明：

(1) “类型说明符”决定了数组中可存放数据的类型。

(2) “数组名”和变量名相同，必须遵循标识符的命名规则。数组的每一个元素都可以看作一个独立的变量，用数组名和下标来表示。C语言中规定：每个数组第一个元素的下标固定为0，称为下标的下界；最后一个元素的下标为元素个数减1，称为下标的上界。（不要界错！）

(3) “常量表达式”代表的是数组元素的个数，也就是数组的长度。它必须是无符号整型常量，不允许是0、负数和浮点数，也不允许是变量。

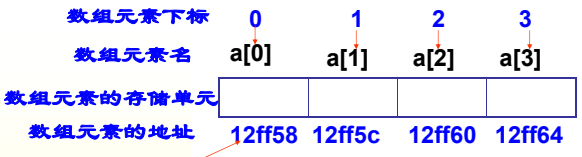
(4) 数组的定义可以和普通变量的定义出现在同一个定义语句中。

例如：　float　k,x[5],y[20];

即在定义单精度变量k的同时，定义了两个单精度型的一维数组x和y。数组x共有5个元素，下标使用范围是0~4；数组y共有20个元素，下标使用范围是0~19。

(5) 数组元素按顺序存储，地址连续。

数组a中的元素，其地址是连续的，如a[0]的地址为12ff58，则从第一个元素a[0]到最后一个元素a[3]的地址为：12ff58，12ff5c，12ff60，12ff64。数组名是常量，其值等于数组的首地址，即首个元素的地址。数组名指向首个元素（类似于指针，后面介绍）。



int a[4];

用数组优点：

- 表述简洁，可读性高
- 便于使用循环结构

例如，以下各个数组的定义是正确的：

```
#define M 20
int a[10];
char b[7+4];
double c[M*2];
```

而以下各个数组的定义是错误的：

```
int n;
int d1[n], d2[n+3]; /* n 是变量 */
char e[-4]; /* 数组元素个数不能是负数 */
double f[2.5]; //数组元素个数不能是小数
float k<12>,m{5};//只能为方括号，其他的错
```

●一维数组的总字节数可按下式计算：

sizeof(类型)* size(数组长度)＝总字节数

方括号内的size是一个正整型的常量表达式，用以说明数组中有多少个成员。

例如：

```
int a[10]; //10个成员，总字节数为40
printf("a=%d\n", sizeof(a));
```

● 一维数组元素的引用

C语言规定数组不能以整体形式参与各种运算。参与各种运算的只能是数组元素，即在程序中不能引用整个数组而只能逐个引用数组元素。

一维数组元素的引用形式为：

数组名[下标]

其中下标可以是整型常量，整型变量或整型表达式。

数组元素引用举例

```
int  a[10],n=2;
/*可以引用元素的从a[0]到a[9]*/
a[5]=6;    a[7]=a[n]++;
a[2]=3;    a[0]=a[5]+a[7]-a[n*3];
```

从中可以看出，在引用数组元素时，下标可以是整型变量，或表达式。

数组元素与普通变量的表现形式不同，但实质是相同的，它也是一种变量。因此，一个数组元素可以象普通变量那样参与赋值、算术运算、输入和输出等操作。

```
#include <stdio.h>  //p65_1
void main( )
{  int  a[10],i,x;
   for(i=0;i<10;i++)a[i]=0;
   scanf("%d",&x);
  while( x ){  a[x%10]++;
               x/=10;  }
  for(i=0;i<10;i++)
       if(a[i])printf("%d = %d\n",i,a[i]);
}
```

```
#include <stdio.h>  //p65_1
void main( )
{
    int  a[10],i;
    char c;
  for(i=0;i<10;i++)a[i]=0;
  while( scanf("%c",&c )) //while( c=getchar() )
      if(c >= 48 && c<='9')a[c-48]++;
      else break;
      for(i=0;i<10;i++)
        if(a[i])printf("%d = %d\n",i,a[i]);
}
```

```
#include <stdio.h>  //p65_6
void main( )
{
  int  a[20],i,x;
   for(i=0;i<10;i++)a[i]=0;
   scanf("%d",&x); //输入整数
   for(i=0;x%10;i++,x/=10)a[i] = x%10 ;
   for(i--;i>=0;i--)
       printf("%d%c",a[i],i?' ':'\n');
}
```

【例4.3】

在数组的某个下标(k)位置插入一个元素。


```
int a[20],  i,k,n,x;
      ……
for(i = n-1; i >= k; i--)
  a[i+1]=a[i];/*自a[n-1]开始逆序至a[k]逐一后移*/
 a[k]=x; /*x插入到k位置*/
 n++;    /*数组的元素增加了一个*/
```

【例4.4】

将数组的某个下标位置(k)的元素删除


```
for(i = k+1; i<n;i++)
   a[i-1]=a[i];   /*自a[k]开始至a[n-1]逐一前移*/
 n--; /*数组的元素减少了一个*/
```

引用小结：

- 数组名[下标]，称之为元素；
- 下标运算符：[ ]；
- 下标范围：0 ~ N-1，N为数组长度；
- 下标可以是常量/变量/表达式；
- 超出下标范围的数组元素称之为越界，会引起不可预料之后果！因为C语言并不检验数组边界。

● 一维数组的初始化

数组的初始化是指在定义数组的同时为数组元素赋初始值。

一维数组在定义时进行初始化的格式为：

类型说明符 数组名[常量表达式] =
{值1，值2，…，值n}；

其中，大括号中的各个值依次对应赋给数组中的各个元素。各个值之间用逗号隔开。

♦给全部元素赋初值 ：

```
int d[5]={0, 1, 2, 3, 4};
```

则有：d[0]=0;d[1]=1;d[2]=2;d[3]=3;
d[4]=4,d[5]=5;

♦如果对数组的全部元素都予以设定初值时，可以不指定数组元素的个数。

例如：

```
int f[]={5, 6, 7, 8, 9};
```

此时由花括号内的初值个数确定数组f有5个元素。

此例等价为

```
int f[5]={5, 6, 7, 8, 9};
```

♦ 初始化元素的个数小于数组元素个数的情况

1）假定数组的元素总数为N，可以只给前m个元素赋初值（N>m），则后部分没有获得初值的元素则置相应类型的默认值（如int型置整数0，char型置字符’\0’，float型置实0.000000等）。

例如定义：

```
int  x[5]={1,2,3};
```

则有：x[0]=1; x[1]=2; x[2]=3; x[3]=0; x[4]=0。

注意：初值个数不允许超过数组元素的总数

例如以下的定义是错误的：

```
int j[5] = {0, 1, 2, 3, 4, 5};
```

初值个数（=6）超过了定义的数组长度（=5）。

♦除了初始化，不能对整个数组直接赋值，只能逐个赋值：

```
a ={0,1,2,3,4};no!
for(i=0;i<5;i++)a[i]=i;   yes!
```

一维数组编程示例

一维数组的定义及其元素引用的例子。

【例3.27】用数组计算Fibonacci数列

要求计算用递推方法计算的 Fibonacci 数列的前n(n≤30)项。并且每行输出5 个数。

用数组表示的 Fibonacci 数列中的各项为：

```
int fa[30];
fa[0]=0; /*fa0=0*/
fa[1]=1; /*fa1=1 */
fa[i]=fa[i-1]+fa[i-2]; /*fai=fai-1+fai-2*/
```

```
//程序如下:
#include <stdio.h>
#define N 30
void main(){int  fa[N], i;
     fa[0] = 0;fa[1] = 1;
     for(i=2; i<N; i++)
          fa[i] = fa[i-1] + fa[i-2];
     for(i=0; i<N; i++)
     {
          if(i%5 == 0)printf("\n");
          printf("%d\t", fa[i]);
     }
}
```

【例4.5】在数a[]的前n个元素中找值等于变量key值元素的下标。

在数组的前n个元素中找值为key的元素，有多种解法。

(1)数据的顺序查找（方案一）

最直观解法是从数组的第一个元素开始，顺序查找至数组的末尾，若存在值为key的元素，程序就终止查找过程；若不存在值为key的元素，程序将找遍整个数组。按此想法编写的程序代码如下：

```
for(i = 0; i < n; i++)
  if(key == a[i]) break; /* 找到终止循环*/
    /* 在这里，若找到i < n; 否则i = n */
```

(2)数据的顺序查找（方案二）。不用 break 语句的程序代码：

```
for(i=0;i<n&&key!=a[i];i++) ;
/* 无论是否找到，都将结束循环 */
/* 在这里，若找到i < n; 否则i = n */
```

假定每个元素的查找机会均等，则采用上述查找方法，查找的平均次数v为：

$$v = (1 + 2 + 3 + \ldots + n)/n = (n+1)/2$$

若在数组中没有指定值的元素，则需查找n次。

(3)数据的顺序查找（方案三：设置哨兵）。

首先将 key 复制到第n+1 元素(a[n])（如数组定义时已预留了空闲的元素）的位置，称为设置了一个哨兵，然后从第一个元素开始顺序寻找，这能简化寻找循环的控制条件。

```
a[n] = key;
for(i = 0; key != a[i]; i++);
/* 若找到 i < n; 否则 i = n */
```

这种写法，数组需多用一个元素，但程序比前一种更简单。

(4)数据的顺序查找（方案四：二分查找法）。

假定数组a的元素已按它们的值从小到大的顺序存放(有序)。则二分法是很好的查找方法。

其算法基本思想是：对任意a[i]到a[j](i<=j)的连续一段元素，根据它们值的有序性，试探位置m=(i+j)/2上的元素,a[m]与key比较有三种可能结果：

如果：key = a[m]；找到，结束查找。

如果：key > a[m]；下一轮的查找区间为 [m+1, j]。

如果：key < a[m]；下一轮的查找区间为 [i, m-1]。

当 j<i 时，区间[i, j]变为一个空区间，即表示在数组a中没有值为key的元素。由于每轮查找后，使查找区间减半，因此称此查找方法为二分法查找。显然，初始查找的区间为i=0, j=n-1。

二分法查找可用C代码描述如下：

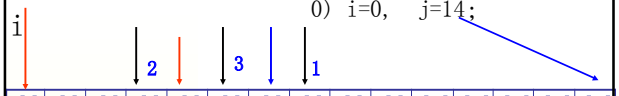
```
i = 0 ;   j = n-1 ; /* 设定初始查找区间 */
while(i <= j) {  // while循环
  m = (i+j)/2 ;
  if (key == a[m])break;//查找成功，结束循环
     else if (key > a[m])
          i = m+1; /* 改变区间，继续查找 */
        else
          j = m-1; /* 改变区间，继续查找 */
}
/* 找到时，i <= j, a[m] == key; 否则 i > j */
```

```
for(i=0, j=n-1; i<= j; )//for循环
{   m = (i+j)/2 ;
    if(key == a[m]) /* 查找成功，结束循环 */
    {
        printf("%d\n", m);/*输出a[m]的下标m */
        break;
     }
     else if(key > a[m])
        i = m + 1; /* 改变区间，继续查找 */
        else
          j = m - 1; /* 改变区间，继续查找 */
}
```

二分法算法演示

例 1：已知n=15, key=36

0) i=0, j=14;



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]
2	4	7	11	27	36	38	40	42	49	50	52	61	68	72

1) m=(0+14)/2=7, a[7]=40>key, j=m-1=6;

2) m=(0+6)/2=3, a[3]=11<key, i=m+1=4;

3) m=(4+6)/2=5, a[5]=36=key, 找到，退出。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]
2	4	7	11	27	36	37	38	42	49	50	51	52	56	72

例 2：已知n=15, key=45

0) i=0, j=14;

1) m=(0+14)/2=7, a[7]=38<key, i=m+1=8;

2) m=(8+14)/2=11, a[11]=51>key, j=m-1=10;

3) m=(8+10)/2=9, a[9]=49>key, j=m-1=9;

4) m=(8+9)/2=8, a[8]=42<key, i=m+1=9;

5) i>j, 查找失败，返回n=15，结束。

【例4.6】按递增顺序生成集合M 的前n 个元素。(p72)

集合 M 定义如下：

（1）整数1 属于M；

（2）如果整数X 属于M，则整数2X+1 和3X+1 也属于M；

（3）再没有别的整数属于M。

根据集合 M 的定义，它的前几个元素的枚举过程为

M = {1, …}

=> M = {1, 3, …} 2 x 1 + 1 = 3

=> M = {1, 3, 4, …} 3 x 1 + 1 = 4

=> M = {1, 3, 4, 7, …} 2 x 3 + 1 = 7

=> M = {1, 3, 4, 7, 9, …} 2 x 4 + 1 = 9

首先，为存储M 的元素设计一个足够大的整数组m，用j 表示集合M 中已生成的元素个数。按题意，首先将1放入集合M 中，然后按递增顺序用M 中的现有元素枚举出M 的新元素。

【算法分析和设计】

设定义数组 m[N]来存储M 集合（N足够大）。由于每个元素都可能施行两种枚举操作，因此需要两个指针e2 和e3，用以分别指向即将执行2*m[e2]+1 和3*m[e3]+1枚举操作基数的位置（元素下标）。另外，由于M 是按递增顺序排列的，所以需要对它们即将执行枚举产生的基数进行比较大小，

小的先枚举产生M 的新元素。而每枚举一次，e2 或者e3 需要加1。

因此，算法描述如下：

初始，令 m[0]为1，e2 和e3 均为0，当前M中元素的个数j为1，集合的容量为n。

如果 j < n，进行循环：

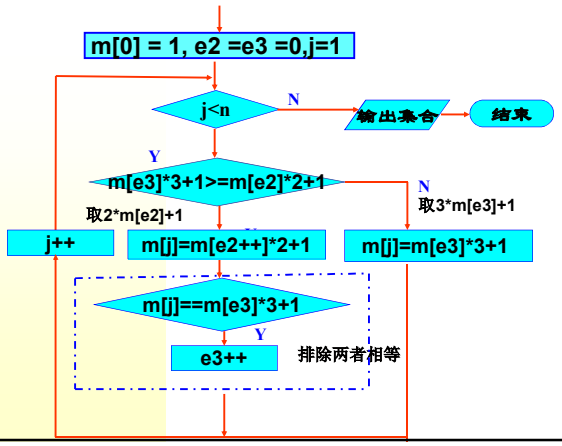
比较 2*m[e2]+1 和3*m[e3]+1：

如果 2*m[e2]+1 较小，则令m[j]=2*m[e2]+1，e2 和j 均加1；

如果 3*m[e3]+1 较小，则令m[j]=3*m[e3]+1，e3 和j 均加1；

否则表示两者相等，任取其一，e2、e3 和j 均加1。

否则，表示 j>=n，结束循环。



【生成集合M 的程序】

```
#include <stdio.h>
#define N 100
void main()
{
    short m[N]={1}, n, e2=0, e3=0, j=1;
    scanf("%d", &n); //生成n个元素
    while(j<n)
    {
```

```
        if(m[e3]*3+1>=m[e2]*2+1)
        { //对m[e2]执行2*x+1枚举操作
            m[j]=m[e2++]*2+1;
            if(m[e3]*3+1==m[j])
                e3++;//当两个元素枚举值相同时,同时枚举
        }
        else
            m[j]=m[e3++]*3+1; //对m[e2]执行2*x+1枚举操作
        j++;
    }
    for(j=0; j<n; j++)
        printf("%d\t", m[j]);
    printf("\n");
}
```

【生成集合 M 的改进程序】

```
#include <stdio.h>
#define N 1000
void main()
{       short m[N], n, e2, e3, j, m2, m3;
        scanf("%d", &n); //生成n个元素
        for(m[0]=j=1, e2=e3=0; j<n; j++)
        {
            m2 = 2 * m[e2] + 1;   //计算m2 和m3
            m3 = 3 * m[e3] + 1;
            m[j] = m2 < m3 ? m2 : m3; // 从m2 和m3 中选择小的送m[j]
            e2 += m2 <= m3;     // 更新e2
            e3 += m2 >= m3;     // 更新e3
        }
        for(j=0; j<n; j++)
            printf("%d\t",m[j]);
        printf("\n");
}
```

【例4.7】从含n个整数的数组中，找出其中出现次数最多且是最先出现的整数。（自学）

【解题思路】

设数组为a[]，为了找出a[]中出现次数最多且最早出现的元素，令变量pos存储数组中出现次数最多的元素的下标，c是该元素在数组中出现的次数(初值为0)。程序用循环顺序考察数组的每个元素，对当前正在考察的元素a[i]，用循环统计出数组中与a[i]等值的元素个数tc。统计结束后，让tc与c比较，如果tc的值大于c，就用tc更新c，并用i更新pos；否则，不更新。

```
for(c = i = 0; i < n-1; i++) //c:记录最多次数
{
   for(tc=1,j=i+1;j<n;j++)//统计a[i]出现次数
      if (a[j] == a[i])
         tc++;//记录相等值出现的次数
   if (tc > c)
   { /* 找到出现次数更多的整数 */
      c = tc;    pos = i; //保留最大数和它的位置
   }
}
printf("出现最多的值是%d 出现的次数是 %d\n",
                        a[pos],c);
```

【例4.8】对数组作整理，使其中小于0的元素移到前面，等于0的元素留在中间，而大于0的元素移到后面。

例如：

{5, -4, 0, -2, 7, 0, 8}

将整理为

{-4, -2, 0, 0, 7, 8, 5}

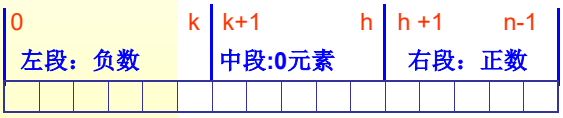
【算法分析和设计】

引入变量k、h，在整理过程中，把数组分为三段”

左段 0 ~k (a[0]~a[k])中存放负数元素，

中段k+1 ~h(a[k+1] ~a[h]) 存放0元素，

右段h+1 ~n-1 (a[h+1]~a[n-1])中存放正数元素。



根据元素a[j]小于0、等于0和大于0这3种情况，分别有3种处理办法:

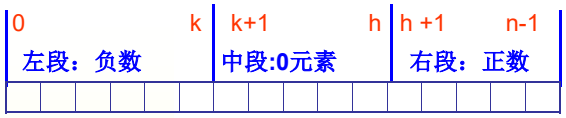
(1) a[j]<0：交换 a[j] 和 a[k]；令k++，小于0的元素多了一个，j++，准备考察下一个元素。

(2) a[j]=0：等于0的元素多了一个，令j++，准备考察下一个元素。

(3) a[j]>0：交换 a[j] 和 a[h]；令h--，大于0的元素多了一个。由于交换前的a[h]是还未考察过的元素，所以j不能增1。

初始时,令 j = 0， k = 0 , h = n-1 。

然后令j从0 开始向右搜索,直至h 为止：



【算法演示】

{5, -4, 0, -2, 7, 0, 8} 将整理为 {-4, -2, 0, 0, 7, 8, 5}

0	1	2	3	4	5	6	
{5	-4	0	-2	7	0	8}	a[j]>0,a[j]⇔a[h],h--,h=5
j,k						h	
{8	-4	0	-2	7	0	5}	
j,k					h		a[j]>0,a[j]⇔a[h],h--,h=4
{0	-4	0	-2	7	8	5}	
k	j			h			a[j]=0,j++;j=1,a[j]=-4<0
{0	-4	0	-2	7	8	5}	
k	j,			h			a[k]⇔a[j],
{-4	0	0	-2	7	8	5}	k++,k=1,j++,j=2
	k,	j		h			a[j]=0,j++;j=3,a[j]=-2<0
{-4	0	0	-2	7	8	5}	
	k		j	h			a[k]⇔a[j],
{-4	0	0	-2	7	8	5}	k++,k=2,j++,j=4
		k		j,h			
{-4	-2	0	0	7	8	5}	

【例4.8】数组整理程序:

```
#include <stdio.h>
#define MAXN 1000
void main()
{   int j, n, k, temp, h,a[MAXN];
    printf("Enter n\n");
    scanf("%d", &n); // 输入数组中实际数据
    printf("Enter a[0] --a[%d]\n", n-1);
    for(j = 0; j < n; j++)scanf("%d", &a[j]);
```

```
  j = k = 0; h = n-1;
  while (j <= h)
     if (a[j] < 0) { temp=a[j]; a[j]=a[k]; a[k]=temp;
                     j++; k++;
                   }
     else if (a[j] == 0) j++;
     else { temp=a[j]; a[j]=a[h]; a[h]=temp;
            h--;
     }
  for(j = 0; j < n; j++)printf("%4d", a[j]);/* 输出结果*/
  printf("\n\n\n");
}
```

排序算法

排序算法是计算机算法中最常用的算法之一。

排序是指对一个数据序列进行处理，使得该序列中的数据按照一定规律按次序排列。最简单的排序是整数的升序或者降序排列。

例如，序列{8, 4, 3, 6, 9, 2, 7}的升序排序为{2, 3, 4, 6, 7, 8, 9}。

排序有许多种算法，例如插入法、选择法、冒泡法、二分法、希尔法、等等。

本教材使用的是冒泡法。

【例4.10】输入n个整数，对它们用冒泡法从小到大排序，然后输出。

冒泡排序：一种类似于轻者上浮、重物下沉的排序算法，将相邻元素进行比较，小者置前，大者放后，不断依次比较，直至最终结果。

冒泡排序算法的基本思想是对数据序列进行 n-1 次循环。

具体做法是：对数组作多次比较调整遍历，在每次循环中，认为数据序列中的 n 个数据是未排序的。依次对相邻两个数据进行比较，如果较大数在前，则将两数交换(设从小到大排序)。在一次循环结束后，最前面一个数据必然是最小的。

方案一：如：int a[]={ 7,8,4,5,1 };

第1 次遍历：

7 8 4 5 1
7 8 4 1 5
7 8 1 4 5
7 1 8 4 5
1 7 8 4 5

第2 次遍历：

1 7 8 4 5
1 7 8 4 5
1 7 4 8 5
1 4 7 8 5

第3 次遍历：

1 4 7 8 5
1 4 7 5 8
1 4 5 7 8

第4 次遍历：

1 4 5 7 8
1 4 5 7 8

其中，下划线表示要交换的元素，红体字表示已排序的元素。

```
/* 冒泡排序算法描述 */
for(i = 0;i < n-1;i++)// 控制n-1次比较调整遍历
    for(j= n-1;j>i;j--) /* 需要比较 n-1-i 次 */
      if(a[j]<a[j-1]){  /* a[j]与a[j-1]交换 */
          temp   = a[j];
          a[j]   = a[j-1];
          a[j-1] = temp;
       }
```

结论：对于N个元素的数组：

- 遍历次数：N-1；
- 第i轮遍历的比较次数：N-1-i；
- 总比较次数：N*(N-1)/2

方案二：冒泡法存在的问题

观察对数据序列{1, 7, 8, 4, 5}进行冒泡排序的过程

第1 次遍历：

1 7 8 4 5
1 7 8 4 5
1 7 4 8 5
1 4 7 8 5
1 4 7 8 5

第2 次遍历：

1 4 7 8 5
1 4 7 5 8
1 4 5 7 8
1 4 5 7 8

第3 次遍历：

1 4 5 7 8
1 4 5 7 8
1 4 5 7 8

第4 次遍历：

1 4 5 7 8
1 4 5 7 8

在第二次循环后，实际上已经完成排序，后两轮可以不做了。但由于冒泡法中外循环的次数是固定的，对数据序列{1, 4, 5, 7, 8}进行冒泡排序时，虽然该数据序列已排序，冒泡法还是要作n-1次循环。从例中得出，对于一个数据序列，如果存在某些子序列是已排序的话，应该可以节省循环的次数的。

根据这种情况，在冒泡排序过程中，如果某次遍历未发生交换调整情况，这时数组实际上已排好了序。程序若发现这种情况后，应提早结束排序过程。

为此，程序引入一个起标志作用的变量，每次遍历前，预置该变量的值为0。当发生交换时，置该变量的值为1，一次遍历结束时，就检查该变量的值，若其值为1，说明发生过交换，继续下一次遍历；如该变量的值为0，说明未发生过交换，则立即结束排序循环。相应的排序代码改为：

冒泡法排序优化片断

```
for(i=0;i<n-1;i++)
{    //控制 n-1 次比较调整遍历
   for(flg=0,j=n-1;j>i;j--)/* 比较 n-1-i 次 */
      if(a[j] < a[j-1])
      {
          temp   = a[j]; a[j]   = a[j-1];
          a[j-1] = temp;
          flg  = 1;//有元素交换的情况
      }
      if (flg == 0) break;
 }
```

方案三：自己分析

更好的改进。在冒泡排序过程中，如果某次遍历在某个位置j发生最后一次交换，以后的位置都未发生交换，这说明以后位置的全部元素都是已排好序的。则后一次遍历的上界可立即缩至上一次遍历的最后交换处之前。例如，数列a[] = {1, 2, 3, 6, 4, 7, 5}，第一次遍历，j在6处发生元素交换，j在4处发生交换，j在3、2、1处都未发生元素交换。最后交换处是j为4。

这次遍历使序列变成：

a[] = {1, 2, 3, 4, 6, 5, 7}

a[] = {1, 2, 3, 4, 6, 5, 7}

下一次遍历的范围的上界可立即缩短至上一次遍历的最后交换处之前，即位置5。

程序为利用这个性质，引入变量up，每次遍历至up之前，另引入变量k记录每次遍历的最后元素交换位置。为了考虑到可能某次遍历时一次也不发生元素交换的情况，在每次遍历前，预置变量k为n。一次遍历结束后，赋k给up，作为下一次遍历的上界。冒泡排序过程至up为n结束，up的初值为0，表示第一次遍历至首元素。

```
up = 0;  /* 一次遍历至up之前，初始时，遍历至
            首元素之前 */
while (up < n) {    /* 还有遍历范围时循环 */
for(k = n, j = n-1; j > up; j--) /* 比较至up */
    if(a[j] < a[j-1]) { /* a[j] 与 a[j-1] 交换 */
        temp   = a[j];  a[j]   = a[j-1];
        a[j-1] = temp;
        k  = j; /* 记录发生交换的位置 */
    }
  up = k;
}
```

【例4.11】设有顺序编号为1至n的n（<100）个人按顺时针顺序站成一个圆圈。首先从第1个人开始，按顺时针顺序从1开始报数，报到第m（<n）个人，令其出列。然后再从出列的下一个人开始，按顺时针顺序从1开始报数，报到第m个人，再令其出列，------ 。

如此下去，直到圆圈不再有人为止。求这n个人出列的顺序。

【解题思路】

程序首先输入n和m，接着将数组的前n个元素顺序设置成1至n，表示n个人按编号顺序站成一个圆圈。然后，求n人的出列顺序。

令a[j]是当前报数者，从第一个人开始报数，j的初值为0。如果a[j]的报数为m，则他的编号a[j]输出，并将a[j]改为0，表示对应的人已出列。让i从0至n-1变化，表示共出列n次。每次出列之前要报数m次，引入k计数报数次数。出列的人不再报数，只有哪些还未出列的人要报数。

报数过程用循环实现，如果a[j]的值为0，表示该人已出列；如果a[j]的值不为0，表示该人还未出列，该人报数，让k增1。如果增1后的k值为m，表示a[j]是这轮报数m的人，这一次的报数循环结束，让a[j]出列。为了表示这n人站成一个圆圈，实现考察下一个人的j增1代码需写成j=(j+1)%n。表示j不等于n-1时，j简单地增1；如果j为n-1，表示最后一个人，这一次增1将对应到第一个人，j变为0。

例如：有10人，报数为3的出局，则有下面的排列

原排列：	1	2	3	4	5	6	7	8	9	10
现排列：	3	6	9	2	7	1	8	5	10	4

```
#include <stdio.h>
#define N 100
int main(){  int i, j, k, a[N], n, m;
     printf("输入n & m\n");scanf("%d%d", &n, &m);
     for(i = 0; i < n; i++)a[i] = i+1;
     for(j = 0, i = 0; i < n; i++) {
          //从第1人开始报数(j = 0)，共有n论报数
        for(k = 0; ;j = (j+1)%n) //寻找出局者
            if(a[j] &&++k == m) break;//找到出局者，退出
      printf("%4d", a[j]);  //打印出局者
      a[j] = 0;   //将出局者置0
   }
   printf("\n程序结束\n");
return 0;
}
```

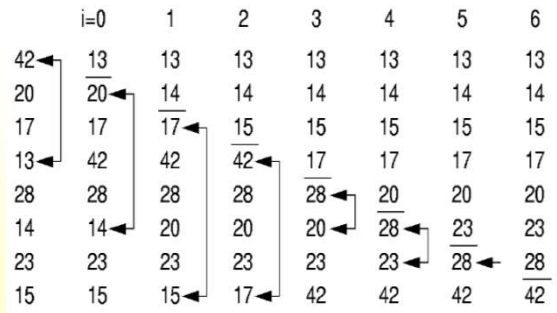
若要将出局者的顺序保留,则可以用下面的方法.

```
#include <stdio.h>
#define N 20
  void main(){int a[N],i,n,m,k,tmp,j;
   printf("输入n & m\n");scanf("%d%d", &n, &m);
   for(i=0;i<n;i++) a[i]=i+1;//初始化,执行n次
   for (i=0, k = n; k > 0; k-- ) //i为起始位置,逐个出局，执行n-1次
   {     i = ( i + m - 1) % k; //第i个出局者位置
         tmp = a[i];         //保留出局者数据
         for(j=i;j<n-1;j++)a[j] = a[j+1];//移出出局者
         a[n-1] = tmp;//出局者交换到第n-1位置
    }for(i=0;i<n;i++)//输出结果
     {     if(i%5==0)printf("\n");//一行输出5个数据
           printf("%d\t",a[i]);
     }printf("程序结束！\n");
}
```

常用的选择排序的算法描述

其算法思想是：

首先从下标为0的元素开始，在数组内找最小元素，并将这最小元素与下标为0的元素交换；接着从下标为1的元素开始，在数组内找最小元素，并将这最小元素与下标为1的元素交换；依次类推，直至从下标为n-2的元素开始，在数组内找最小元素，并将这最小元素与下标为n-2的元素交换。到此，数组排序完毕。



```
void  main()
{int i,n,k,x,min,min_k,a[100];
scanf("%d",&n); /* */
for (i=0;i<n;i++)scanf("%d",&a[i]);/*输入10个整数*/
for (k=0;k<n-1;k++)  /*控制排序n-1步 */
{ min=a[k];   /* 假设第k数是最小值，用min记录*/
  min_k=k;  /* 用min_k 记录当前最小值的下标*/
 for (i=k+1;i<n;i++)/*在a[k]到a[n-1]寻找最小数*/
    if (a[i]<min)  /*成立，记录新的最小数和它的下标*/
      { min=a[i]; min_k=i;}
   x=a[min_k];a[min_k]=a[k];a[k]=x; /*交换最小数和第k个数*/
 }
 for (i=0;i<=n-1;i++) /*输出排序后的n个数*/
      printf("%5d",a[i]);
 printf("\n");  }
```

p94:11题意（自学）：

为了输出p/n(p<n)小数部分的各位数字，可采用逐位输出的办法：

由p*10%n的余数作为新的小数，如此循环，逐步得到p/n的任意位小数。

p0=1,p1,p2...pi...,pj,当某个pi=pj时，自pi，pj求出的各位小数也重复出现，也就是说一个新的循环节开始。即每次用p求新的一位余数时，查p是否在余数表中，如不在，把p存入r[]中，并求出新的小数和调整p，如p已在余数表中，则一个新的循环节开始，结束求1/n各位小数的循环。

举例:1/7

```
      商
7 | 1   ----  1
    3   ----  4
    2   ----  2
    6   ----  8
    4   ----  5
    5   ----  7
    1         结束
```

余数数组

0	
1	1
2	1
3	1
4	1
5	1
6	1
7	
⋮	
n-2	
n-1	

```
#include <stdio.h>
void main()
{       int n,p,i,r[100];    //r为余数数组
        for(n=2;n<51;n++)
        {       for(i=0;i<n;i++)r[i]=0;   //余数数组赋0
                printf("1/%d=0.",n);  //输出1/n
                p=1;
                while(r[p]==0)//查p是否在余数表中,若在,则结束循环
                {       r[p]=1;
                        printf("%d",p*10/n); //输出结果
                        p=p*10%n;       //得新的一位小数
                }
                printf("\n");
        }
}
```