简单构造数据类型

数组的引出

例:某班有30名学生,求该班成绩的平均分

```
#include <stdio.h>
int main(void)
{
    int  i, s, sum = 0 ;
    float  ave ;
    for(i=1; i<=30 ; i++)
    {
          scanf("%d", &s);
          sum=sum+s;
    }
     ave = sum / 30;
     printf("ave=%f", ave);
     return 0;
}
```

这里只使用了一个变量s，虽然通过循环我们输入了30个学生的成绩，但循环结束后s中只是第30个学生的成绩，前面29个学生的成绩都没有保存下来

如果要求保存这30名学生的成绩，最后再输出，应该怎么办?

思考？

要定义30个变量：S0、S1、S2、……、S29 吗？

- 数组：由具有相同类型的固定数量的元素组成的集合
- 数组元素：每一个数组元素都是一个变量，用数组名和下标唯一地确定数组元素。为了与一般的变量相区别，我们称数组元素为下标变量。
- 下标变量在数组中的位置序号称下标。
- 数组中的每个元素属于同一类型。

一维数组的定义

- 格式：类型标识符　数组名 [常量表达式] ;
  - 例：int a[10];
- 说明
  - 数组名要符合合法标识符的命名规则
  - 数组长度指数组中元素的个数，由常量表达式的值确定
  - 数组的类型实际上是指数组元素的取值类型。对于同一个数组，所有元素的数据类型都是相同的。
  - 数组的下标从0开始
    - 例: int a[10]中a有10个元素，所以数组长度为10，数组元素分别是: a[0], a[1] … a[8],a[9]

- 说明（续）
  - 数组名表示了一个存储区的首地址(即第一个数组元素的地址)
    - 例：一个变量x的地址可以用&x来表示；一个数组a的地址就用数组名a来表示， a等价于&a[0]
  - 常量表达式中可包括常量和符号常量，但不能包含变量和实数

```
int  n;
scanf("%d",&n);
int  a[n];
```

```
int  b[8.5];
```

错

```
int  n=6;
int  a[n];
```

```
#define  SIZE  8
…
int  a[2+3];
float  b[SIZE];
```

对

数组元素的引用

- 形式：数组名[下标]
- 说明
  - 下标可以是整型常量或整型表达式
    - 如： a[1] ， a[2*3]
  - C语言规定只能逐个引用数组元素而不能一次引用整个数组
  - 对下标的引用不要超过下标的最大值
    - 例： int a[5], 只能到a[4]

数组元素的引用示例

```
#include  <stdio.h>
int main(void)
{
      int  i, sum, s[30] ;
      float  ave ;
      for(sum=0, i=0; i<=29 ; i++)            //注意下标不要越界
      {
          scanf("%d",  &s[i]);                //逐个引用数组元素
           sum = sum + s[i];
      }
      ave = sum / 30;
      printf("ave=%f", ave);
      for(i=0; i<30 ; i++)
           printf("%d,", s[i]);
      return 0;
}
```

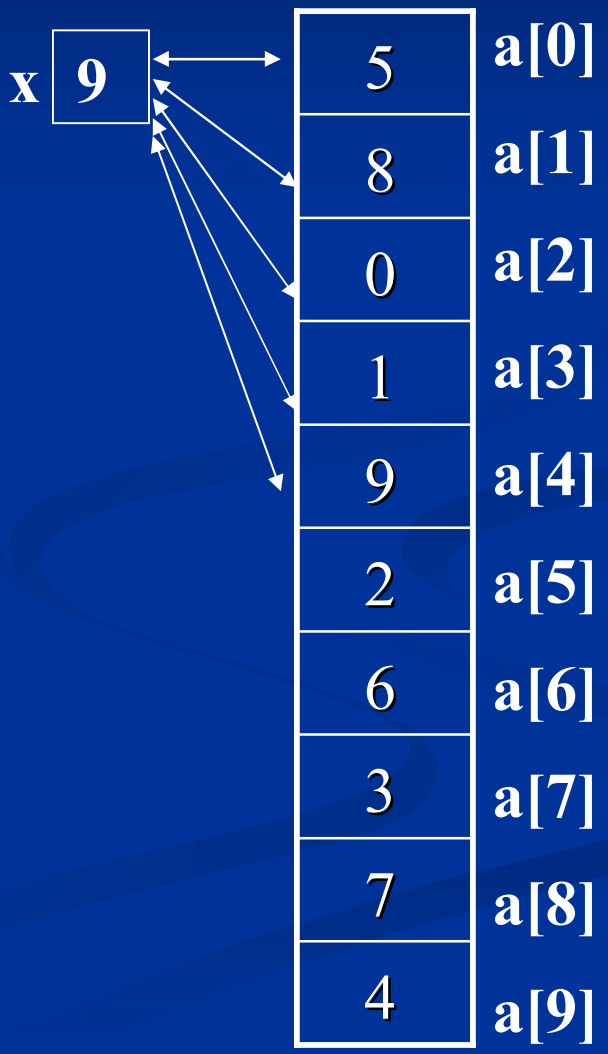
一维数组的初始化

- 概念：在定义一维数组时对各个元素指定初始值称为数组的初始化
  - 例： int a[5] = { 1 , 3 , 5 , 7 , 9 } ;
- 说明
  - 对数组的全体元素指定初值，初值用 { } 括起来，数据之间用逗号分开。这种情况下，可以不指明数组的长度，系统会根据{ }内数据的个数确定数组的长度。
    - 例：int a[ ] = { 1 , 3 , 5 , 7 , 9 } ;
  - 对数组中部分元素指定初值 (这时不能省略数组长度)。
    - 例：int a[5] = { 1 , 3 , 5 };
  - 使数组中的全部元素初始值都为 0。
    - 例：int a[5] = { 0, 0, 0, 0, 0 } ;　　可简写为： int a[5]={ 0 } ;
  - 初始值个数不能大于数组长度。

数组应用示例

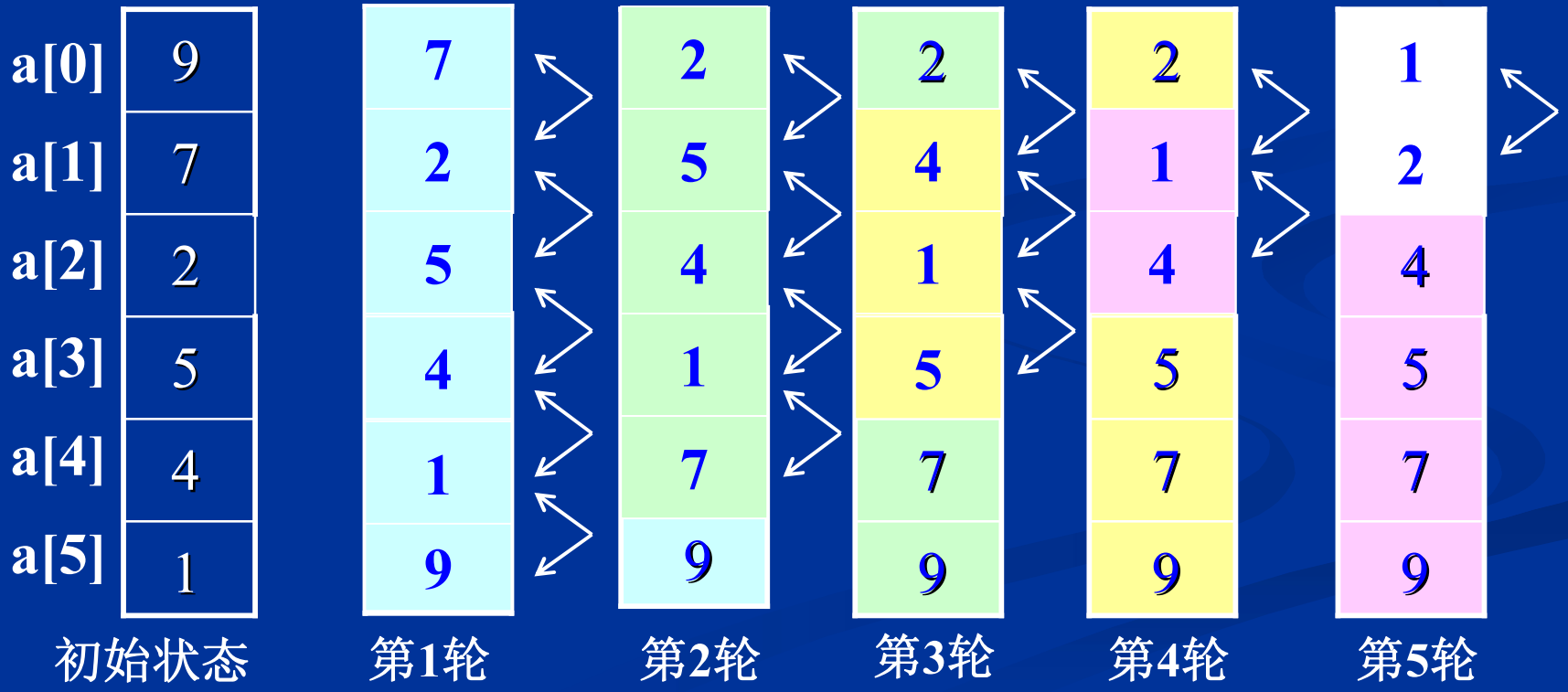
例：输入一个数据，在已知数组中查找是否有该数据。

```
#include  <stdio.h>
int main(void )
{
   int  i , x ;
   int  a[10]={ 5, 8, 0, 1, 9, 2, 6, 3, 7, 4 };
   scanf("%d", &x);
   for ( i=0 ; i<10 ; i++)
        if ( x == a[i] )
        {
             printf("find! a[%d]=x\n", i);
             break;                  //找到后结束
        }
   if ( i==10 )
        printf("no  find!\n");
   return 0;
}
```



冒泡排序法

- 用冒泡排序法对6个数进行排序(从小到大)
- 冒泡排序方法的思想：依次比较相邻的两个数，将小数放前面，大数放后面。n个数排序需要进行n-1轮比较，从第1轮到第n-1轮，各轮的比较次数依次为：n-1次、n-2次 … 1次。



冒泡排序法示例程序

```
#include  <stdio.h>
int  main(void)
{
    int a[6] , i , j , t;
    for ( i=0 ; i<6 ; i++)
        scanf("%d", &a[i] );
    for ( i=0 ; i<6-1 ; i++)
        for ( j=0 ; j<6-1-i ; j++)
            if ( a[j]>a[j+1] )
            {
                t = a[j] ;
                a[j] = a[j+1] ;
                a[j+1] = t ;
            }
    for ( i=0 ; i<6 ; i++)
        printf( "%3d", a[i] );
    return 0;
}
```

输入6个数据

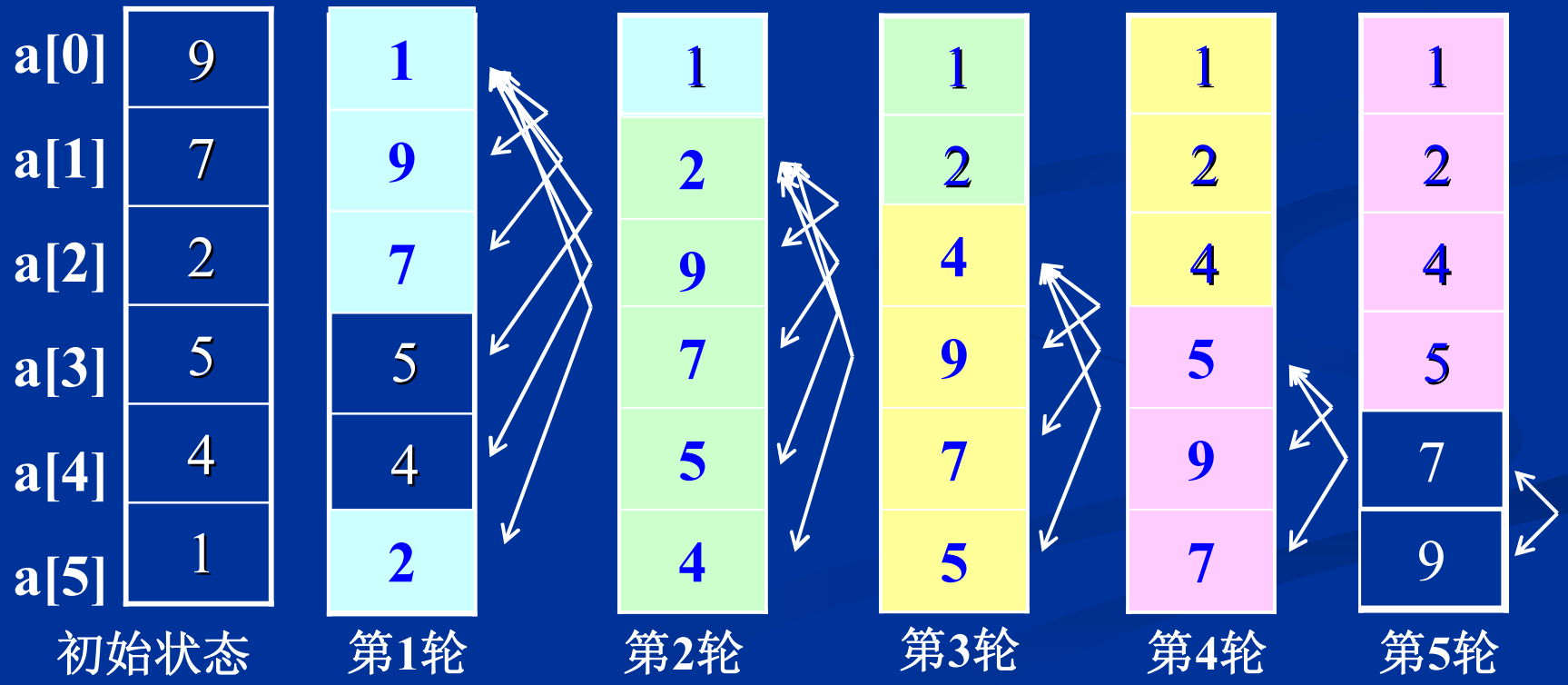
用嵌套的for循环实现排序
外层循环控制进行几轮比较
内层循环控制每一轮的比较次数

若前面的数大于后面的数，则进行交换

输出排序后的6个数据

选择排序法

- 用选择排序法对6个数进行排序(从小到大)
- 选择排序方法思想：第1轮比较时，用a[0]依次与a[1]到a[5]进行比较，如果a[0]较大则进行交换，第1轮结束后，a[0]中为最小数。以后各轮比较过程与第1轮类似。



选择排序法示例程序

```
#include <stdio.h>
int main(void)
{
    int a[6] , i , j , t;
    for ( i=0 ; i<6 ; i++)
        scanf("%d", &a[i] );
    for ( i=0 ; i<6-1 ; i++)
      for ( j=i+1 ; j<6 ; j++)
          if ( a[i] > a[j] )
          {
                t = a[i] ;
                a[i] = a[j] ;
                a[j] = t ;
          }
    for ( i=0 ; i<6 ; i++)
        printf( "%3d", a[i] );
    return 0;
}
```

冒泡排序程序的改进

- 从这道例题中我们发现，在进行完第二轮后，数据就排好序了，在第三轮中数据没有进行一次交换，说明排序已经完成了，第四、五轮的比较都是多余的，这种情况下应该终止排序过程。

	初始状态	第1轮	第2轮	第3轮	第4轮	第5轮
a[0]	9	7	1	1	1	1
a[1]	7	1	2	2	2	2
a[2]	1	2	4	4	4	4
a[3]	2	4	5	5	5	5
a[4]	4	5	7	7	7	7
a[5]	5	9	9	9	9	9

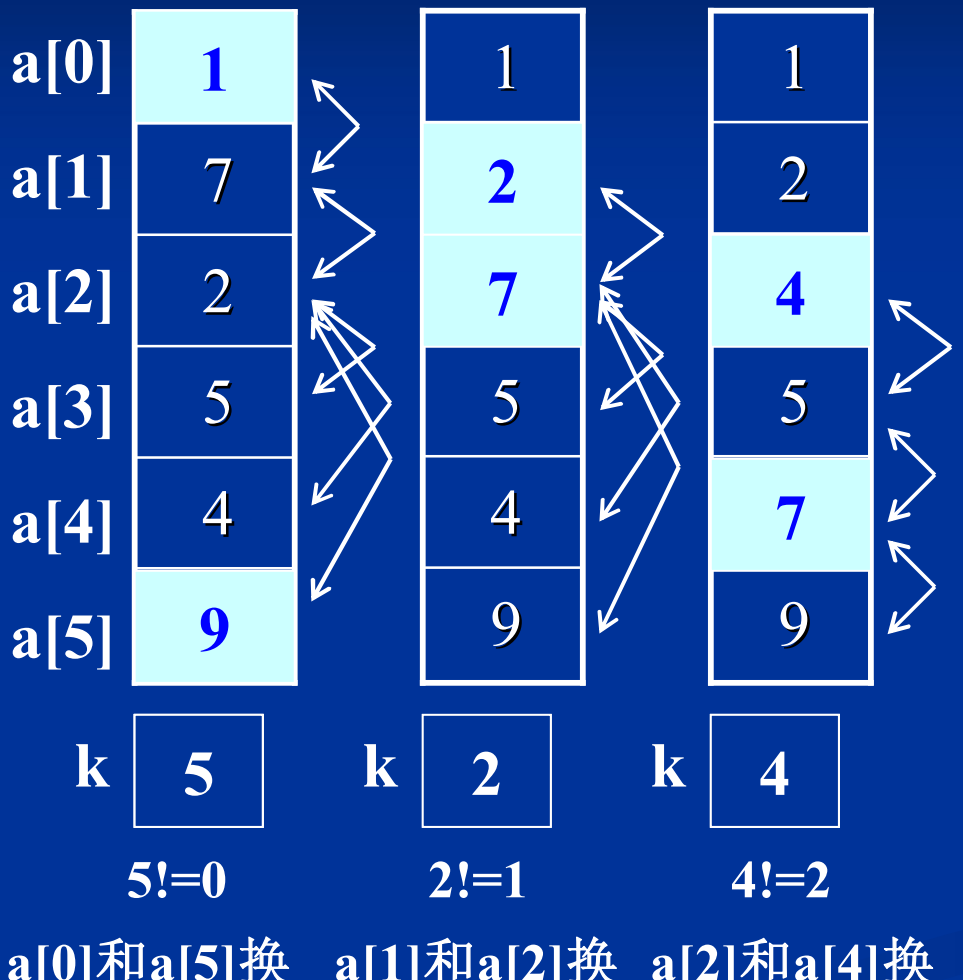
为了解决问题，在程序中设置一个变量flag，用它记录在每一轮比较中是否进行了交换，在每轮比较开始前flag=0，如果在此轮比较中进行了交换，则flag=1，在一轮比较结束后，判断flag的值是否为1，如果值为1，则继续进行排序；如果值为0，说明在此轮比较中没有进行交换(即已完成排序)，此时可终止循环(即结束排序)

```
#include  <stdio.h>
int main (void)
{  int  a[6] , i , j , t , flag;
   for ( i=0; i<6; i++)
       scanf("%d", &a[i] );
   i = 0 ;
   do
   {    flag = 0;
       for ( j=0 ; j<5-i ; j++)
          if ( a[j] > a[j+1] )
          {
                t = a[j] ;
                a[j] = a[j+1] ;
                a[j+1] = t ;
                flag = 1;
          }
        i++ ;
    } while ( flag ) ;
   for ( i=0 ; i<6 ; i++)
        printf( "%3d",a[i] );
   return 0;
}
```

选择排序程序的改进

- 分析选择排序过程发现，在每一轮的比较中交换次数太多，我们可以尽量减少交换次数，实际上每轮比较只要一次进行交换就能完成排序。
- 改进方法：
  - 先从要排序的n个数中找出最小的数，把它放在第一个位置
  - 再从剩下的n-1个数中找出最小的数，把它放在第二个位置
  - 这样重复做下去，即可
- 改进后的选择排序方法数据进行比较的次数并没有减少，但每一轮只进行一次交换，加快了程序运行速度

设置变量k用以存储当前最小数的下标：

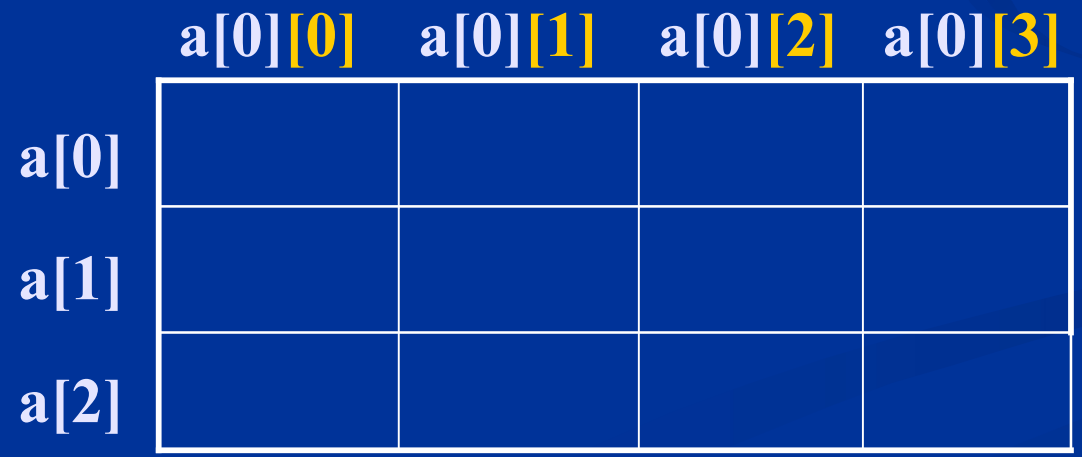



```
#include  <stdio.h>
int main (void)
{
   int a[6] , i , j , k , t;
   for ( i=0 ; i<6 ; i++)
       scanf("%d", &a[i] );
   for ( i=0 ; i<5 ; i++)
   {
       k = i ;
       for ( j=i+1 ; j<6 ; j++)
         if ( a[k] > a[j] )
               k = j ;
       if ( k != i )
       {
            t=a[i] ;
            a[i]=a[k] ;
            a[k]=t ;
       }
   }
   for ( i=0 ; i<6 ; i++)
       printf( "%3d", a[i] );
   return 0;
}
```

二维数组

- 概念：一个一维数组，它的每一个元素都是类型相同的一维数组，就形成一个二维数组。
- 形式：

  类型标识符 数组名[常量表达式1][常量表达式2]
- 例： int a[3][4] ;



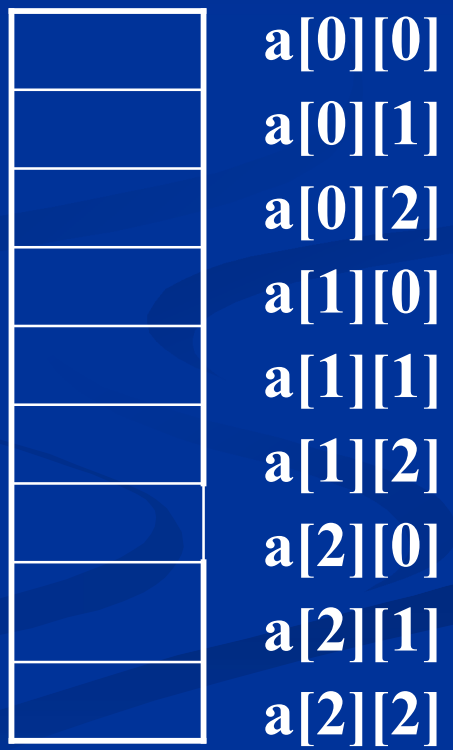
二维数组的存储

- 存储形式：数组的元素在内存中是连续存放的
- 例：int a[3][3]；数组a的存放形式如下：

a[0][0]→a[0][1]→a[0][2]

a[1][0]→a[1][1]→a[1][2]

a[2][0]→a[2][1]→a[2][2]



二维数组的引用

- 形式：数组名[下标1][下标2]
- 例：float a[3][4]; 引用数组元素时应为：

  a[0][0], a[0][1], ……， a[2][3]
- 注意：
  - 每个下标都要用 [ ] 括起来，如a[2][1]不能写成 a[2,1]
  - 下标不要超过定义的范围

二维数组的初始化

- 按行初始化：
  - 例：int a[3][4]={{ 1, 2, 3, 4 }, { 5, 6, 7, 8 }, { 9, 10, 11, 12 }};
  - 将第1对{ }内的数据赋给第1行数组元素，以此类推

1	2	3	4
5	6	7	8
9	10	11	12

- 按数据的排列顺序初始化
  - 例：int a[3][4]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
  - 将数据依次赋给元素 a[0][0] , a[0][1] …a[2][3]

- 对数组的部分元素初始化

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

1	2	0	0
3	0	0	0
0	4	0	0

```
int  a[3][4]={1,2,3,4,5,6};
```

1	2	3	4
5	6	0	0
0	0	0	0

- 对数组的全部元素赋初值时，可以省略第一维的长度，系统会根据数据的个数和第二维的长度自动求出第一维的长度，但不可省第二维的长度。

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

数组a第一维长度为 3

```
int  b[ ][2]={ 1, 2, 3, 4, 5, 6, 7, 8 } ;
```

数组b第一维长度为 4

- 例：找出矩阵中最大的数，并输出其行号和列号

```
#include  <stdio.h>
int main(void)
{
    int  i , j, row=0, col=0, max ;
    int  a[3][4]={{5,2,0,9}, {3,7,12,6}, {10,4,1,8}};
    max = a[0][0];
    for ( i=0; i<3; i++ )
        for ( j=0; j<4; j++ )
            if ( a[i][j] > max )
            {
                    max = a[i][j] ;
                    row = i ;
                    col = j ;
            }
    printf("max=%d\n", max);
    printf("max=a[%d][%d]\n", row , col);
    return 0;
}
```

5	2	0	9
3	7	12	6
10	4	1	8

输出：

max=12

max=a[1][2]

max	12
row	1
col	2

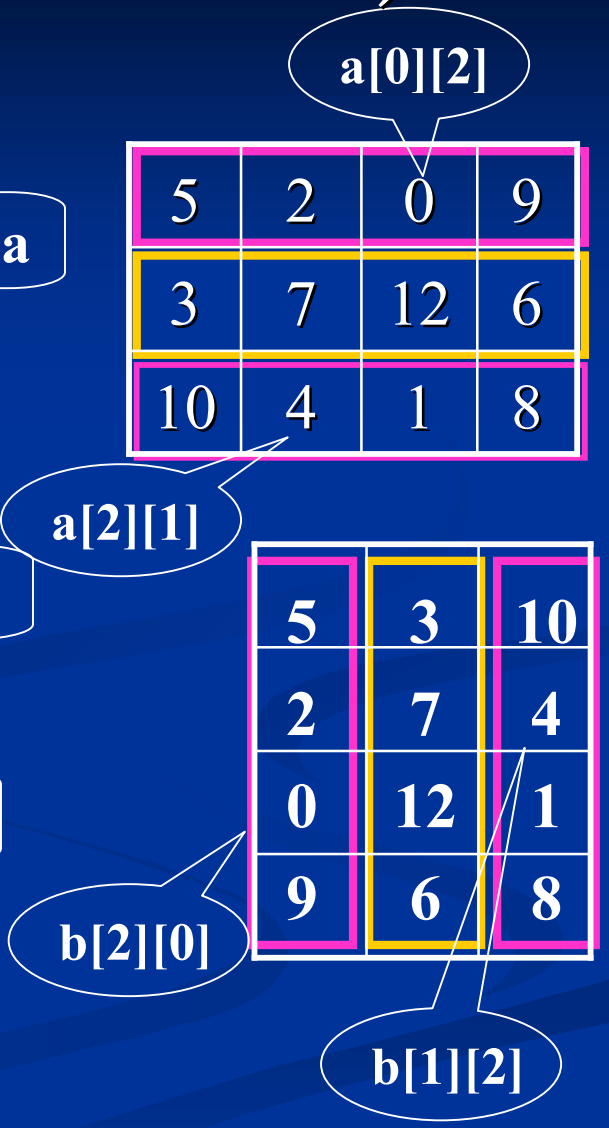
- 例：将一个矩阵进行转置(即原来的行变为列)

```
#include  <stdio.h>
int main(void)
{
    int  a[3][4], b[4][3] , i , j ;
    for ( i=0 ; i<3 ; i++ )
        for ( j=0 ; j<4 ; j++ )
            scanf("%d", &a[i][j] ) ;
    for ( i=0 ; i<3 ; i++ )
        for (j=0 ; j<4 ; j++)
            b[j][i] = a[i][j];
    for ( i=0 ; i<4 ; i++ )
    {
        for ( j=0 ; j<3 ; j++ )
            printf("%5d", b[i][j]) ;
        printf("\n");
    }
    return 0;
}
```

输入数组a

进行矩阵转置

输出数组b




Class is over