高级语言程序设计

第06章　数组

内容提要

数组

- 数组的概念
- 一维数组
- 二维数组
- 数组的常用算法

查找、插入、删除、排序等

数组的概念

构造类型

构造类型是由基本数据类型重新组合而产生的新的类型。

基本数据类型

整型、实型、字符型、枚举型

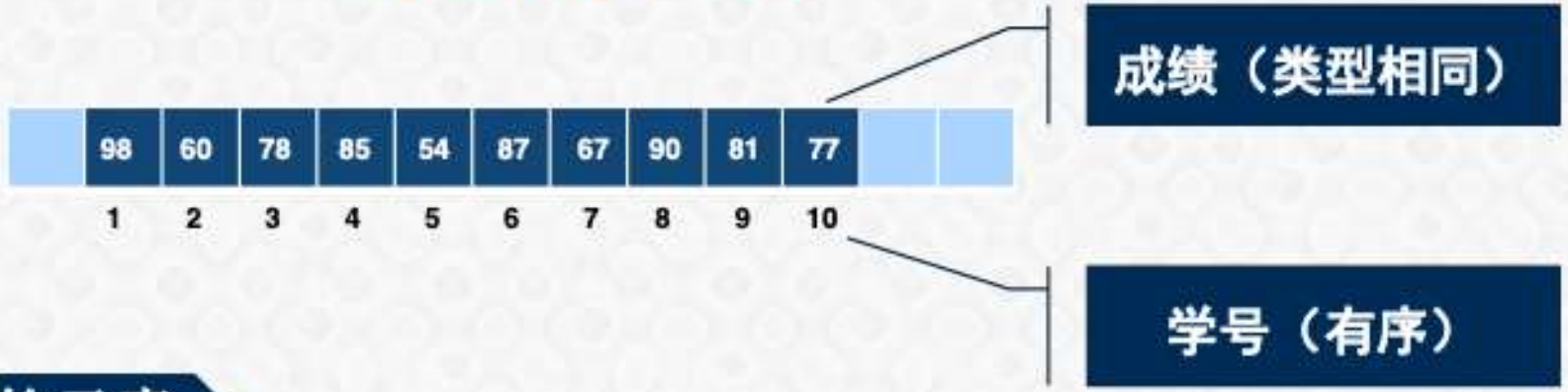
数组的作用

- 问题：
  - 输入3个学生的c语言成绩，计算平均分。

数组

数组的概念

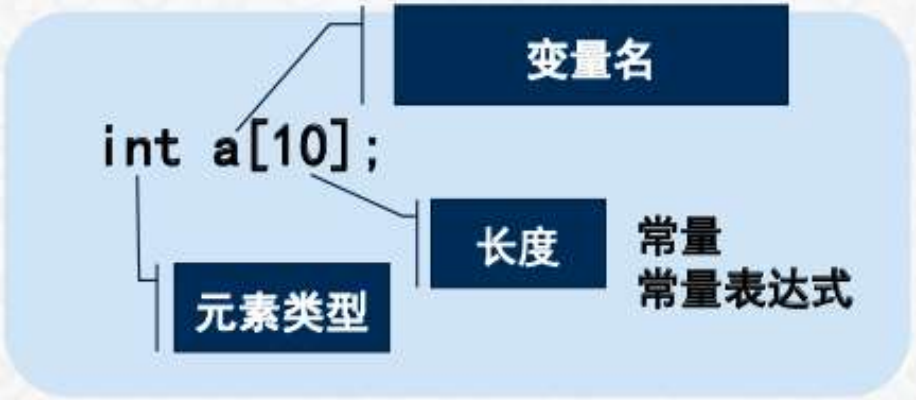
数组是一组**类型相同**的**有序**数据的集合。



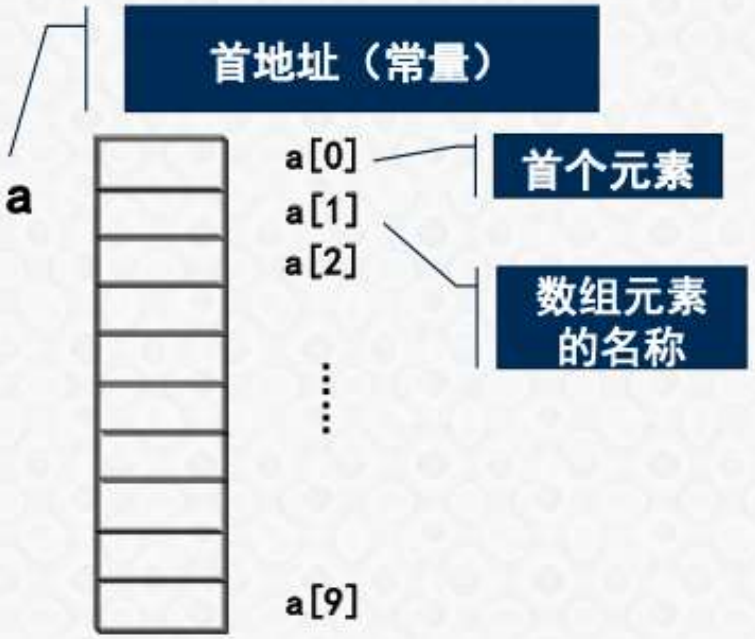
数组的元素

组成数组的变量称为该数组的元素。

6.1 一维数组的定义



我们定义了一个长度为10的整型数组a，数组的每个元素都是整型。



一维数组所占内存空间

- 分配给数组的存储空间是由**数组元素的类型**和**个数决定**的。

```
int a[10];
```

一维数组所占的字节数
=sizeof(元素类型) ×数组长度

所占内存空间： 4*10=40字节

一维数组的初始化

数组的初始化：

在定义数组时，为其全部或部分元素指定初值。

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

相当于：
a[0]=1;a[1]=2;a[2]=3;
a[3]=4;a[4]=5;

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

相当于：
int a[5]={1,2,3,4,5};
数组长度省略

一维数组的初始化

初始化时，没有被赋值的元素的值均为0。

例子：

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

部分赋值，
相当于a[5]={1, 8, 0, 0, 0};

```
int a[5];
```

注意！当不做初始化时所有数组元素的值均为随机数而不是0

```
int a[5]={0};
```

所有元素均为0

错误的初始化示例

例子：

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

对数组元素初始化时只能从左到右依次赋值，只能缺省最右边的元素值。

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

初始值个数不能超过数组定义的长度！

设置

1、下列错误的一维数组初始化是_____。

A `int arr[4]={1,2};`

B `int arr[4]={1, ,3,4,};`

C `int arr[4]={0};`

D `int arr[ ]={1,2,3,4,5};`

提交

2、有定义int a[3]={1,2,3};则sizeof(a)、sizeof(a[0])、sizeof(&a[1])的值分别是（　）

A 4,4,4

B 12,4,4

C 4,1,2

D 12,1,4

提交

设置

3、有定义int a[10];并且已知a代表的地址值为004FFD3C，则&a[6]的值为（ ）

A 004FFD54

B 004FFD50

C 004FFD58

D 004FFD4C

0、1、…、9、A(10)、B(11)、C(12)、D(13)、E(14)、F(15)

a[0] = xxx3C
a[1] = xxx40
a[2] = xxx44
a[3] = xxx48
a[4] = xxx4C
a[5] = xxx50
a[6] = xxx54

提交

一维数组的访问

访问数组：

实际上是指访问数组中的元素。

数组的元素可以通过数组名+下标来访问

数组元素的表示 a[i]

a表示数组的首地址，是个常量；

这里i可以理解为与a[0]之间相隔元素的个数；

方括号。

一维数组的遍历

数组的遍历

访问数组通常要访问数组中的每个元素，这个过程也称为“数组的遍历”，如果要遍历数组中的每个元素，我们可以是借助于循环语句来实现的。

```
for (i=0; i<10;i++)
        scanf( "%d" ,&a[i]);

for (i=0; i<10;i++)
    printf( "%d " ,a[i]);
```

- 例6_1:50个学生成绩的存储与处理

应用举例---数列

例6_2：Fibonacci数列

时间(月)	小兔子(对)	大兔子(对)	兔子总数(对)
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
…	…	…	…

- 算法分析：
- 1) 有一对小兔子，第2个月长成大兔子，长到第3个月开始每个月就生一对小兔子
- 2) 求出1, 2, 3, …n, 这n个月每个月兔子的总对数？

应用举例---数列

例6_2：Fibonacci数列

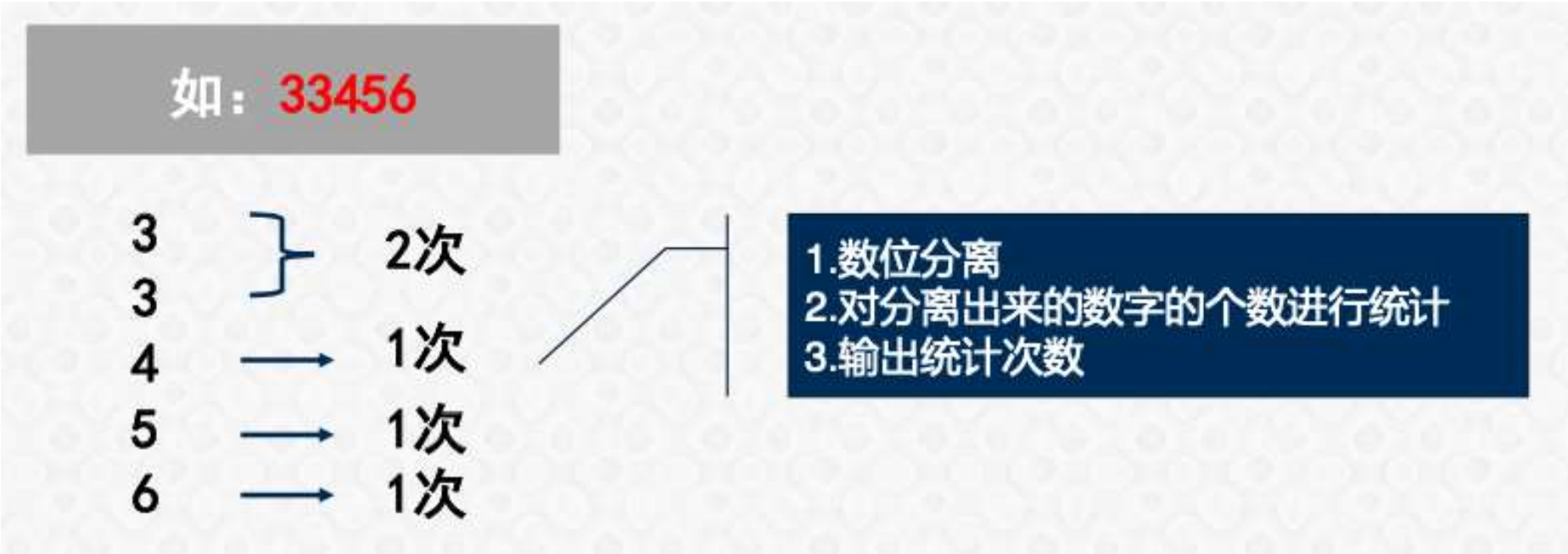
时间(月)	小兔子(对)	大兔子(对)	兔子总数(对)
1	1	0	1
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21
…	…	…	…

$$fib(i)=\begin{cases}1 & i=1 \text{ or } i=2\\ fib(i-1)+fib(i-2) & i\geq 3\end{cases}$$

- 当i=1或2时，Fibonacci数列的值为1；
- 当i>=3时，它的值为前两项的和；

应用举例—数位统计

- 例6_3：统计一个整数中各个数字出现的次数。



应用举例—数位统计

- 例6_3：统计一个整数中各个数字出现的次数。
- 算法分析：
  - 数位分离：while、数组下标统计方法

如果这个数超过3位，采用的方法叫做“斩尾法”。

```
int digit[10]={0};
 while(m)
 {
     i=m%10;
    digit[i]++;
     m=m/10;
 }
```

数组下标的巧用

向函数传递一维数组

- 对使用频率较高的功能，用函数加以模块化（判断质数）
  - 避免重复书写
  - 调用函数（函数首部三要素）
    - 参数传递和返回值，完成函数之间的交流
- 简单变量作为参数传递给函数
- **一维数组作为参数**传递给函数
  - 借助数组完成**批量数据的传递**

向函数传递一维数组

- 数组名a是整个**数组的首地址**（首个元素存放的地址），后面的元素依次存放
  - 已知数组首地址、数组的有效长度，就可以找到数组中的每个元素

➢ 假设定义 了一个长度为10的整型数组a。

```
int a[10];
```

长度

a

首地址

首个元素

a[0]
a[1]
a[2]
⋮ ⋮
a[9]

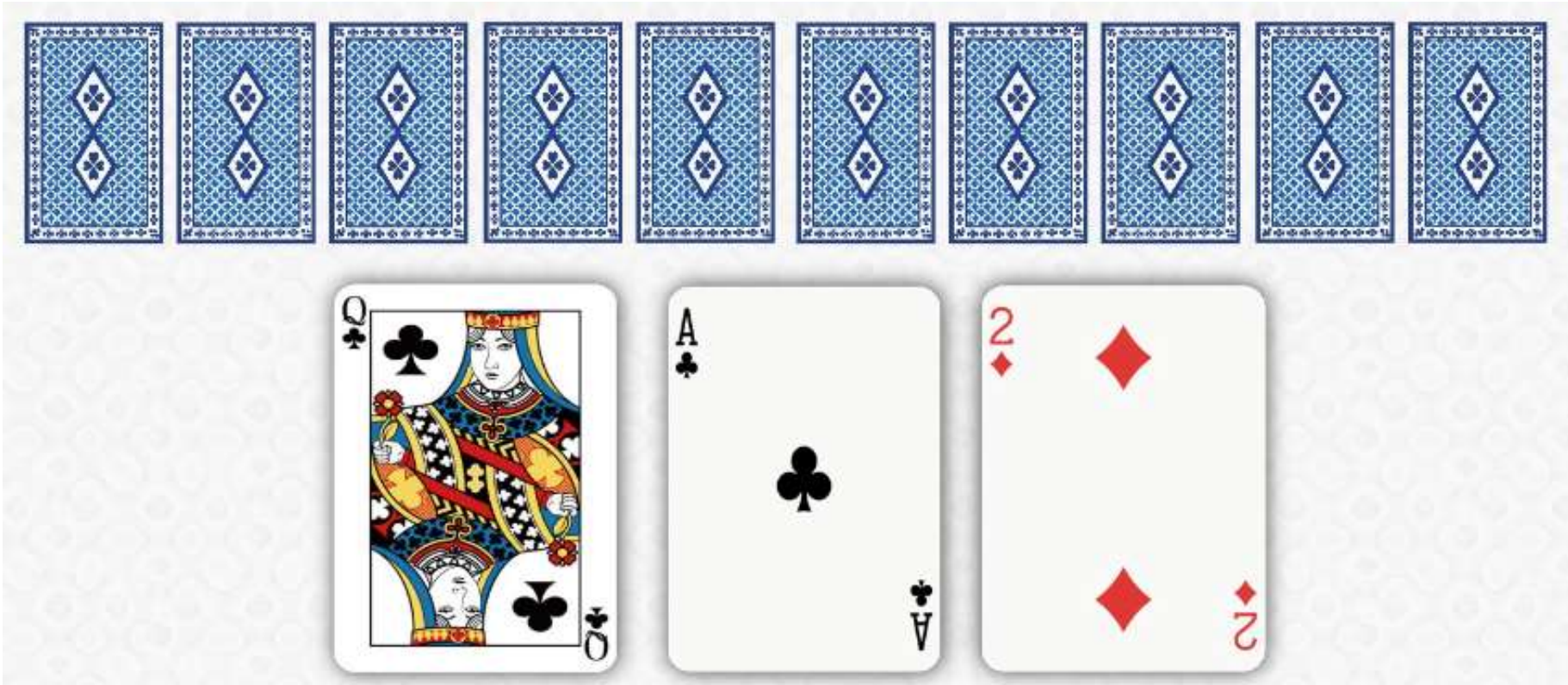
➢ 一般向函数传递一维数组时，参数有两个：**数组的名字和长度n**。

应用举例：最值求解

- 例6_4：用函数完成数组中最大值的求解
- 数据类型：一维数组存储待比较的数据
- 算法：最值的求解

应用举例：最值求解

- 例6_4：用函数完成数组中最大值的求解
- 算法：最值的求解



应用举例：最值求解

- 例6_4：用函数完成数组中最大值的求解
- maxnum算法：开始假设a[0]是最大值，然后依次从a[1]到a[n-1]做比较，当max比当前值a[i]小，则替换max为a[i]
- 循环结束时max中一定存放的数组a中最大值
- 函数返回max给主函数

```
int maxnum(int a[],int n)
{
    int i,max;
    max=a[0] ;
   for (i=1;i<n; i++)
      if (a[i]>max)
            max=a[i];
   return max;
}
```

应用举例：最值求解

- 例6_4：用函数完成数组中最大值的求解
- 数据类型：一维数组存储待比较的数据
- 算法：函数之间批量数据的共享

```
#define N 10
int maxnum(int a[],int n);
int main()
{  int array[N],n;
   int max;
 ……
 max=maxnum(array,N);

……
}
```

```
int maxnum(int a[],int n)
{
    int i,max;
   max=a[0] ;
  for (i=1;i<n; i++)
     if (a[i]>max)
           max=a[i];
  return max;
}
```

地址传递是双向传递，形参的改变会影响实参的变化。

应用举例：最值求解

- 用数组名传递（将数组首地址传过去）
  - 实参的首地址array赋值给形参int a[]（一维数组的长度在做形式参数时省略）
  - 数组a和array 就指向，同一个数组的首个元素，另外一个参数n传递数组的有效长度
- 被调函数maxnum中的数组a和主调函数中的数组array实际上共用同一段内存空间
- 地址传递是双向传递
  - 形参的改变会影响实参的变化
  - 形式参数a中元素发生变化等价于实际参数array数组中元素也发生了变化

内容提要

- 数组的概念
- 一维数组
- **二维数组**（定义、初始化及访问）
- 数组的基本算法

使用二维数组的场合

例：用一维数组表示30位学生的数学成绩

如：s[24]=95;

表示学号 为24的学生，他的数学成绩是95分 。

有语数外三门课的成绩，该学生的数学成绩又怎么表示？

使用二维数组的场合

	1（语文）	2（数学）	3（英语）
1	97	87	92
2	92	91	95
		……	
24	91	95	90
		……	
30	90	81	82

s[24][2]=95;

行　　列

数据需要在两个维度上表达其顺序性，就需要用二维数组。如：数学上的矩阵

二维数组的定义

格式：

类型标识符　数组名[整型常量表达式1]　[整型常量表达式2]；

数组元素类型　与C语言标识符的定名规则一致　两个维度

例：int a[2][3]；

下标值分别是2、3表示第一维长度为2，第二维长度为3表示.　位置

a	a[0][0]	a[0][1]	a[0][2]
	a[1][0]	a[1][1]	a[1][2]

数组所占的字节数
=第一维长度*第二维长度
*sizeof(数据类型)

24

二维数组的存储

原则：行优先

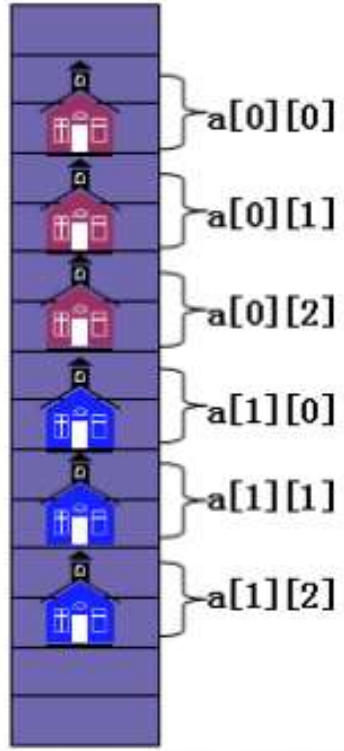
先顺序存放第0行的元素，再存放第1行的元素……

```
int  a[2][3];
```

a[0][0] a[0][1] a[0][2]

a

a[1][0] a[1][1] a[1][2]



二维数组的初始化

原则

按行从左到右依次赋值

例：逐行初始化

```
int a[4][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

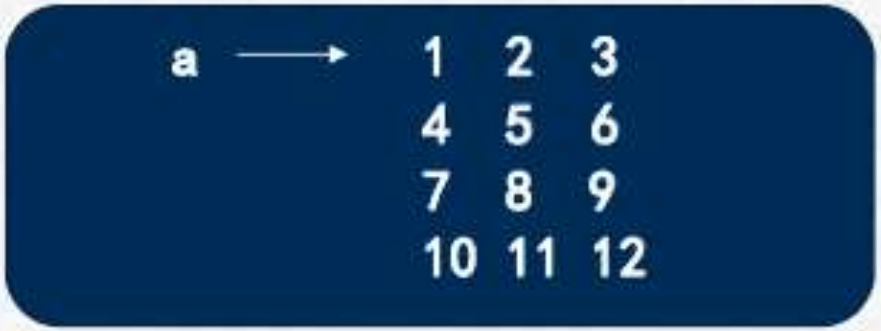
行数　列数

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

二维数组的初始化

- 不分行，用类似一维数组的方式初始化：

```
int a[4][3]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
```



二维数组的初始化

- 全部赋值时，行数可以缺省，列数不能省：

```
int a[ ][3]={{1,2,3},{4,5,6},{7,8,9},{10,11,12}};
```

默认为4

a ⟶

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

二维数组的初始化

- 部分赋值：

```
int a[4][3]={{1,2},{4,5},{7,8,9},{10,11,12}};
```

```
int a[3][4]={1,0,6,0,0,11};
```

a →

1	2	0
4	5	0
7	8	9
10	11	12

- 最简单的初始化： `int a[4][3]={0};`

a →

1	0	6	0
0	11	0	0
0	0	0	0

设置

4、有定义int a[3][3]={{1,2},{3,4,5}}，则元素a[2][1]的值为（ ）

1 2 0
3 4 5
0 0 0

A 3

B 0

C 1

D 2

提交

设置

5、下列哪一个二维数组的初始化是错误的（ ）

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

1 2 3
4 5 6

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

1 2
3 4
5 6

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

1 2 3
4 0 0

D int a[3][2]={0*6};

0 0
0 0
0 0

提交

设置

6、有定义int a[4][3];并且已知数组的首地址为：00FE4B4C，则元素a[2][1]的地址为（ ）

A 00FE4B60

B 00FE4B64

C 00FE4B68

D 00FE4B70

0、1、…、9、A(10)、B(11)、C(12)、D(13)、E(14)、F(15)

[0][0] [0][1] [0][2]
[1][0] [1][1] [1][2]
[2][0] [2][1] [2][2]
[3][0] [3][1] [3][2]

7*4=28
4C+16=5C
5C+4=60
60+8=68

提交

二维数组的访问

- 二维数组元素的表示：数组名+下标

一个下标表示行，
一个下标表示列，
下标值都从0开始。

例： int a[][3]={1, 2, 3, 4, 5, 6, 7}； 请指出元素6的下标值

一个3行3列，
能存放9个元
素的二维数组。

二维数组的访问

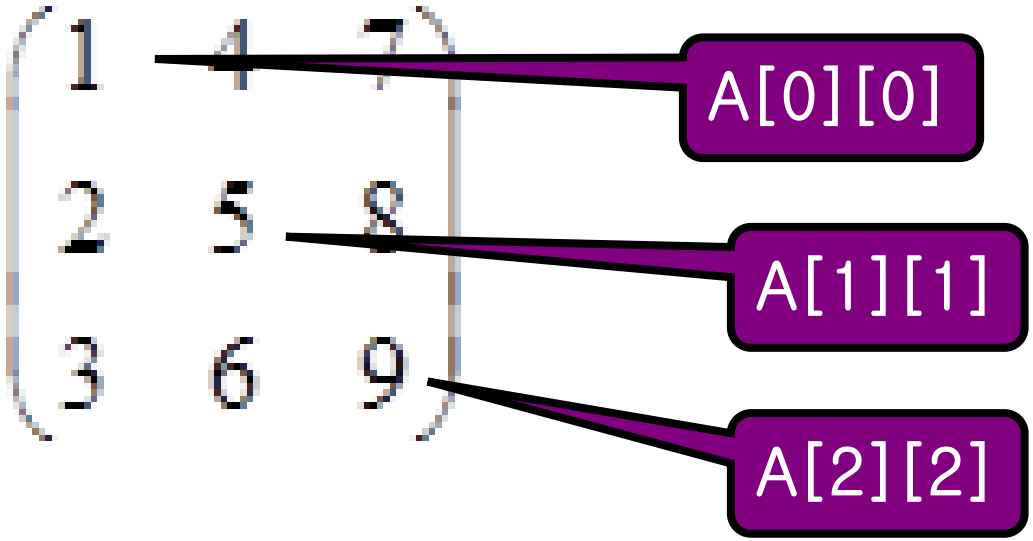
- 二维数组元素的遍历需用两层循环

  外层循环控制行，内层循环控制列

例：
```
int a[3][4],n=1;
for(i=0;i<3;i++)
        for(j=0;j<4;j++)
                a[i][j]=n++;
```

6.2 二维数组

- 例6_5 存储矩阵，并输出对角线



- int A[3][3];

行数　列数

二维数组的访问

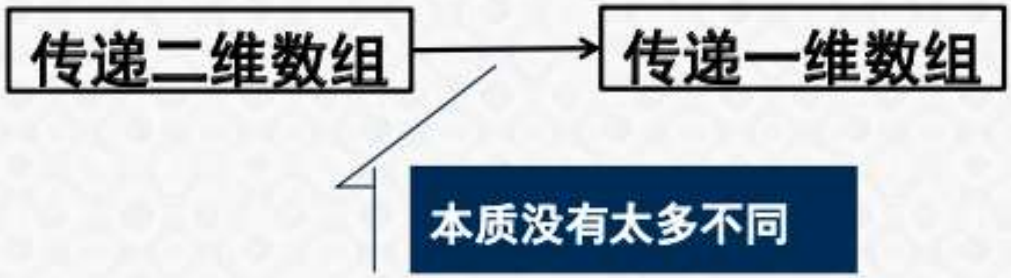
- 例6_6 矩阵转置运算
  - 数据类型:由于矩阵行、列不同，定义两个二维数组分别存放转置前、后的矩阵
  - 算法提示:只需要在控制循环时，先控制列下标再控制行下标就能实现矩阵的转置

例：int a[3][4], b[4][3];

```
for(int i=0;i<4;i++)
    for(int j=0;j<3;j++)
        b[i][j]=a[j][i];
```

传递二维数组

- 如何向函数传递二维数组的问题。



- 形式参数变成了二维数组的数组名，两个下标分别表示行数和列数。

数组名作函数参数，本质上仍然是地址传递。

应用举例：九九乘法表

- 例6_8 实现九九乘法表并输出用户要求的格式。
- 关键问题：
  - 九九乘法表的存储：二维数组
  - 显示方式：
    - 全部
    - 下三角
    - .....

提供的菜单选项

- 关注：二维数组作为函数的形参

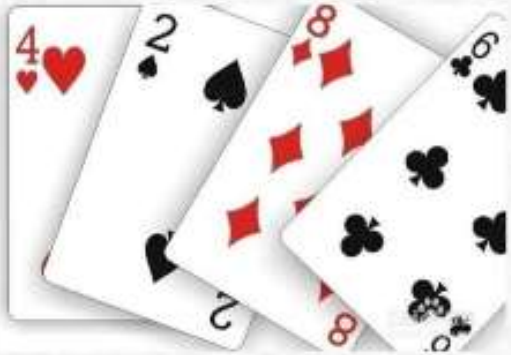
内容提要

二维数组

- 定义、初始化及访问
- 应用

应用举例：模拟发牌

- 模拟扑克牌游戏中的发牌过程。随机将52张扑克牌发给两位玩家。



52张牌的存储

每张牌由“花色”和“数字”决 定。

应用举例：模拟发牌

- 例6_7 模拟扑克牌游戏中的发牌过程，随机将52张扑克发给两个玩家。
  - 黑桃、红桃、梅花、方块，这四种花色。每一种花色都有2、3、4、5、6、7、8、9、10、J、Q、K、A 这13个点数
- 关键问题：
  - 如何表示某张扑克牌？
  - 发牌过程
    - 随机抽出一张扑克牌
    - 标注这张扑克牌属于谁
  - 显示发牌结果

应用举例：模拟发牌

扑克牌信息的存储

花色 \ 数字	2	3	4	5	6	7	8	9	10	J	Q	K	A
	0	1	2	3	4	5	6	7	8	9	10	11	12
♥ 0													
♦ 1				-1									
♠ 2				1		1							
♣ 3													

```
int card[13][4]={0};
```

假设：如果这张牌的存储的值是1，说明是发给玩家1，如果是-1，说明是发给玩家2。

注意的问题

1. 教材中是用13行4列的二维数组，是可以的。

- 用不同的数组存储数据，程序的处理上肯定会有一些不同。所以程序编出来，可能是不一样的，形式上是非常灵活的，但都是实现同一个功能。

注意的问题

2. 问题转化是解题的关键

- 如何将发牌游戏转化为我们计算机可以处理的信息，再运用我们已学习的编程知识来解决问题，这正是我们这题的解题关键 ，也就是问题转化。

注意的问题

3. 多上机实践

- 书上的例题千万不要只看，不去上机试试，一定要先用自己的方法解决一遍，再跟教材中程序进行比较，每一道编程题都不可能只有一种答案，这是也是初学者的感到迷惑和困难的地方。

注意的问题

4. 变量的定义

- 编写程序的时候，对变量的定义不要急于求成，一气呵成，往往在编写程序的过程，根据需要添加变量，注意c程序的变量都定义在函数体内较前位置。

数组常用算法：查找

- 数组的概念
- 一维数组
- 二维数组
- 数组的常用算法
  - **查找**
  - 插入
  - 删除
  - 排序

数组常用算法：查找

- 例6_9 在a数组中查找x，如果存在输出它的下标，否则提示：“Not present!”。
- 算法：
  - 顺序查找法，对数组排列无要求.
  - 查找的过程:从第一个元素开始依次与待查找的元素进行比较，如果相等就查找成功，输出元素及对应下标；如果与所有元素都比较结束仍没有相等元素，则输出元素不存在的提示信息
- 数据结构：
  - 数组a、变量x用于存储要查找的数据
  - 循环控制变量i，循环结束条件：i>=n或者x==a[i]

查找函数的实现

- 接口确定：
  - 功能
  - 入口：数组名、数组长度、待查找的数
  - 返回

```
#define SIZE 10
int main()
 {.....
    //读入array的长度、值和x
   pos=find(array,n,x);
   if(pos<n)
         printf("value=%d, ...);
  else
         printf("Not present!\n");
.......
}
```

```
int find(int a[],int n,int x)
{
   int i=0;
   while(i<n)
  {
        if (x==a[i])
       break;
        i++;
   }
   return i;
}
```

数组常用算法：插入

- 数组的概念
- 一维数组
- 二维数组
- 数组的常用算法
  - 查找
  - **插入**
  - 删除
  - 排序

数组常用算法：插入

➢ 例6.10：将x插入数组a中，要求保持数组非递减有序

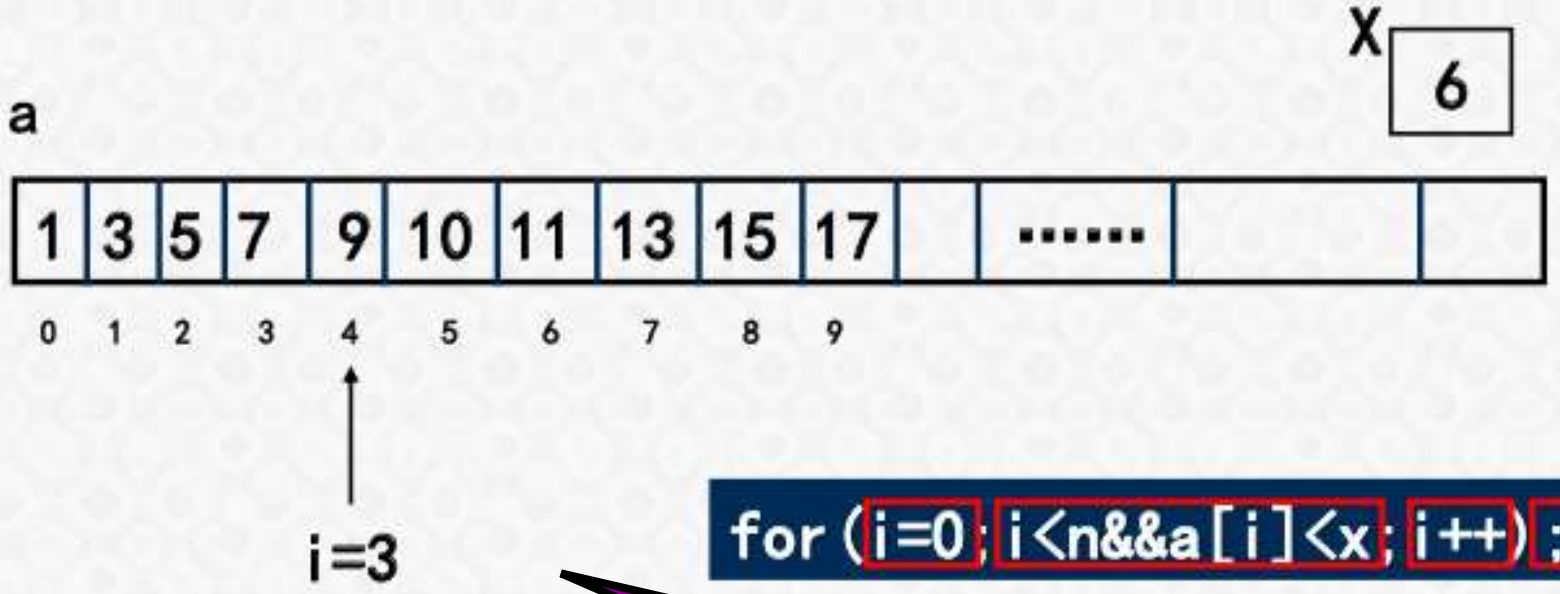
递增，但不是严格递增

- 数组要保持非递减有序

数组本来就是有序的,可以允许相邻两个数相等，x插入之后还要保持原来的递增序列

数组常用算法：插入

- 第一步，要确定待插入点的位置

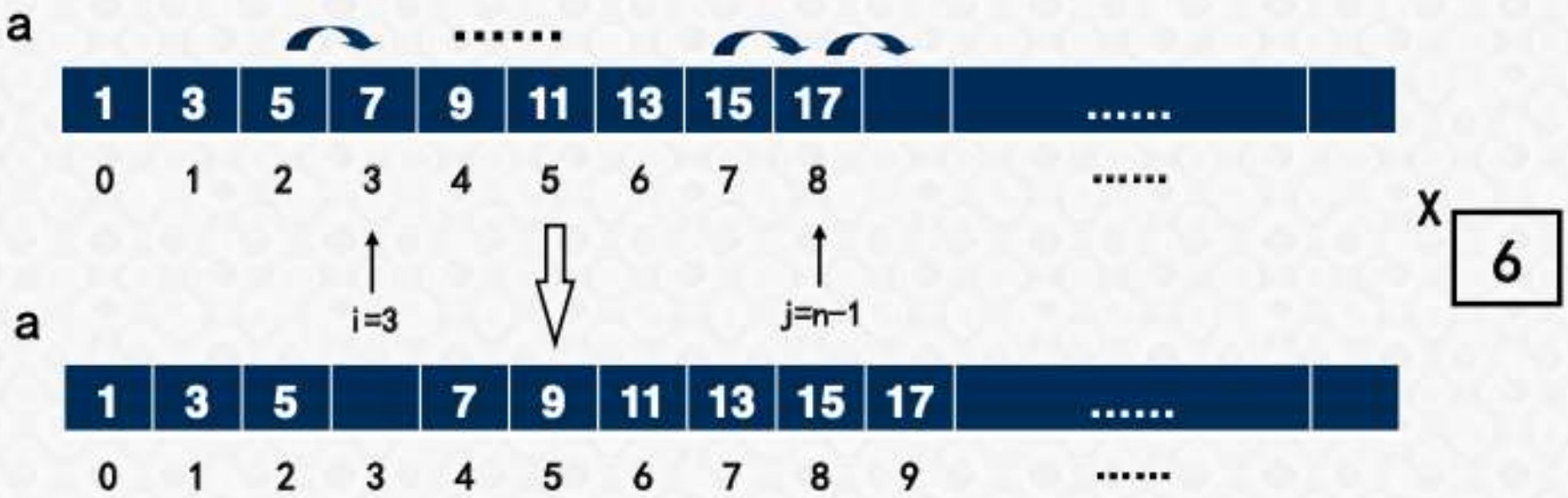



```
for (i=0; i<n&&a[i]<x; i++) ;
```

找到第一个比x大的数(也就是找到7的下标位置)，i 不能超过数据有效长度n

数组常用算法：插入

- 第二步，元素后移，腾出相应位置

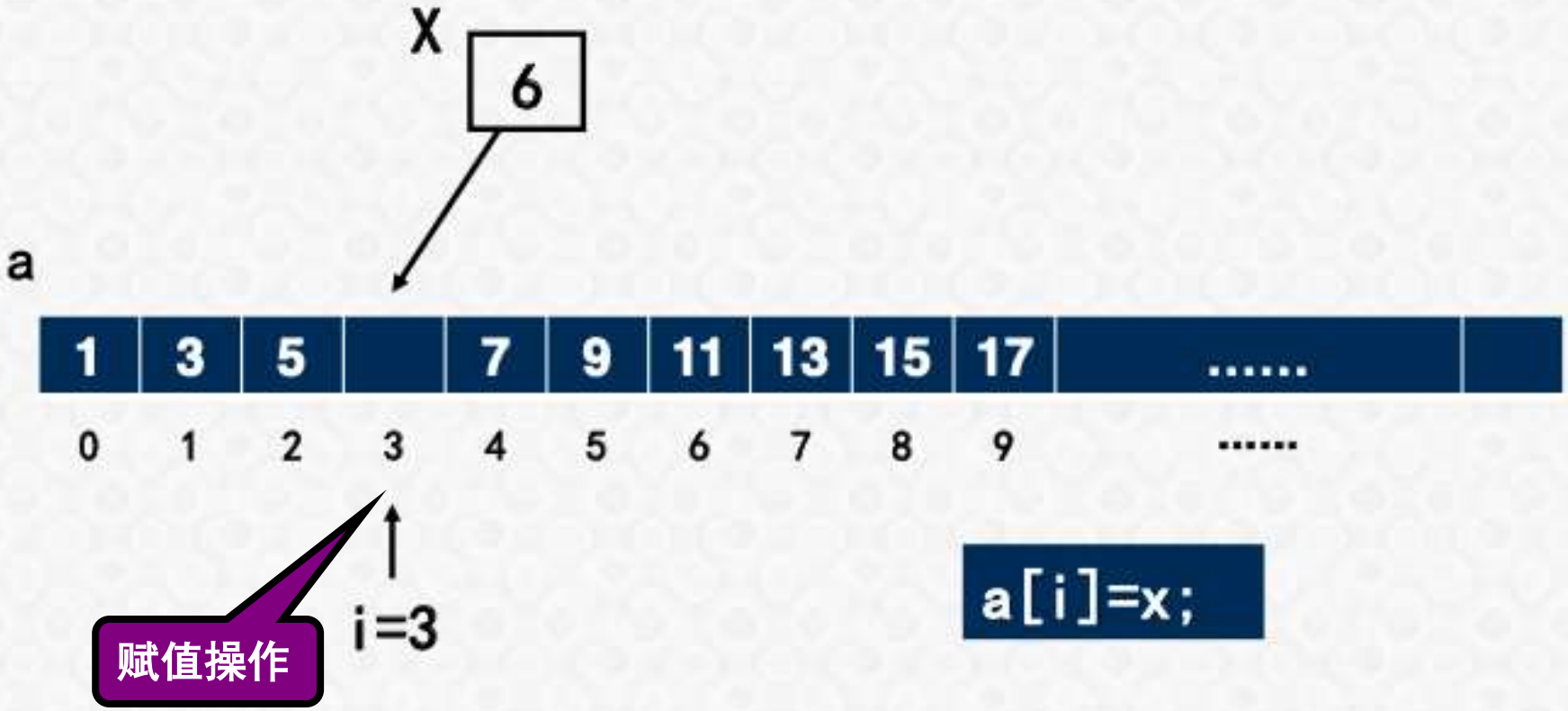



```
for(j=n-1;j>=i;j--)
    a[j+1]=a[j];
```

移动是从最后一个元素开始，不断给后面的元素赋值，直到把元素7移到下标为4的位置上，把下标为3的位置空出来

数组常用算法：插入

- 第三步，在“空”的位置上插入新元素



数组常用算法：插入

- 例6_10 将x插入数组a中保持数组非递减有序
- 算法：有序数组中插入一个数据元素，并保持数组有序。
  - （1）确定待插入位置—循环
  - （2）元素后移，腾出相应位置---后移方法
  - （3）在“空”位置上插入新元素
- 数据结构：
  - 数组a、变量x用于输入要插入的数据
  - 循环控制变量i，两次循环，查找位置以及移动

插入函数的实现

- 接口确定：
  - 功能：有序插入,升序
  - 入口：数组名、数组长度、待插入的数
  - 返回

```
#define SIZE 7
int main()
 {.....
    //初始化array值、读入x
   insert(array,SIZE-1,x);
    .......
}
```

```
void insert(int a[],int n,int x)
{
   for (i=0;i<n&&a[i]<x;i++);
                                  /*定位*/

   for (j=n-1;j>=i;j--)    /*移位*/
      a[j+1]=a[j];

   a[i]=x;                        /*插入*/
}
```

数组常用算法：删除

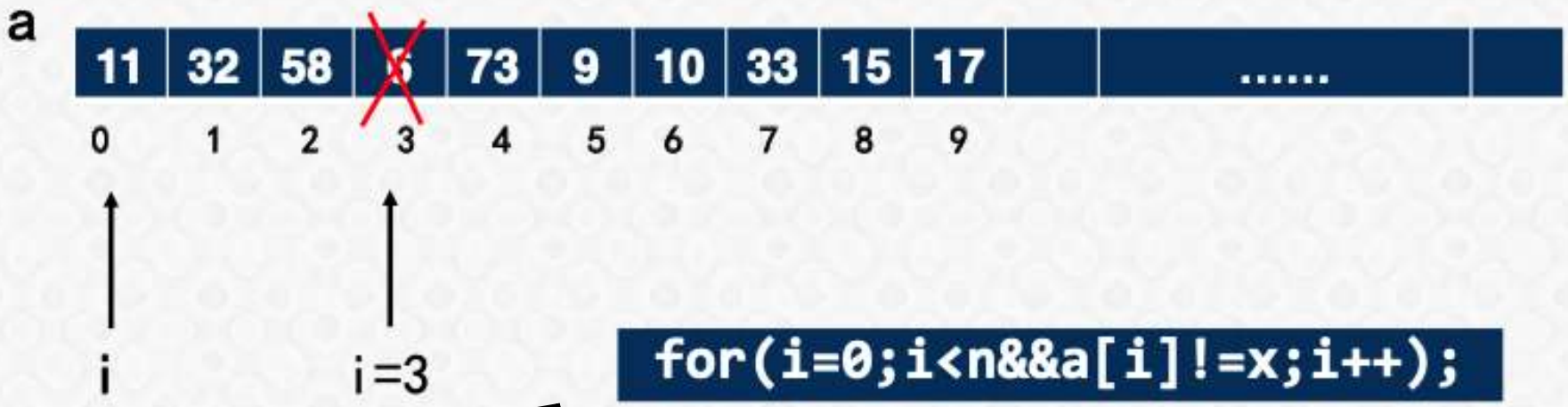
- 数组的概念
- 一维数组
- 二维数组
- 数组的常用算法
  - 查找
  - 插入
  - **删除**
  - 排序

数组常用算法：删除

- 例6.11：从整型数组a中删除第一个等于x的元素，如果x不是数组中的元素，则显示：“can not delete x!”。

数组常用算法：删除

- 第一步，确定待删除元素的位置




```
for(i=0;i<n&&a[i]!=x;i++);
```

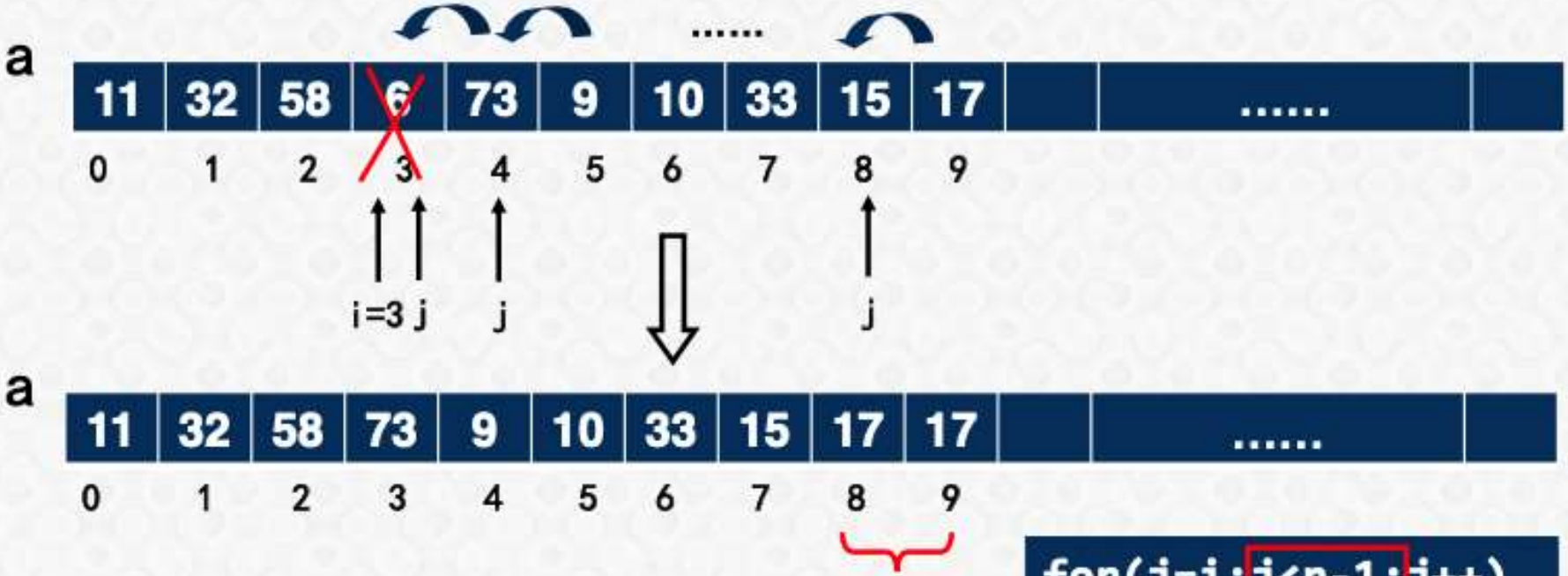
删除：找到第一个等于x

```
for (i=0; i<n&&a[i]<x; i++);
```

插入：找到第一个比x大

数组常用算法：删除

- 第二步，元素依次前移，覆盖待删除元素

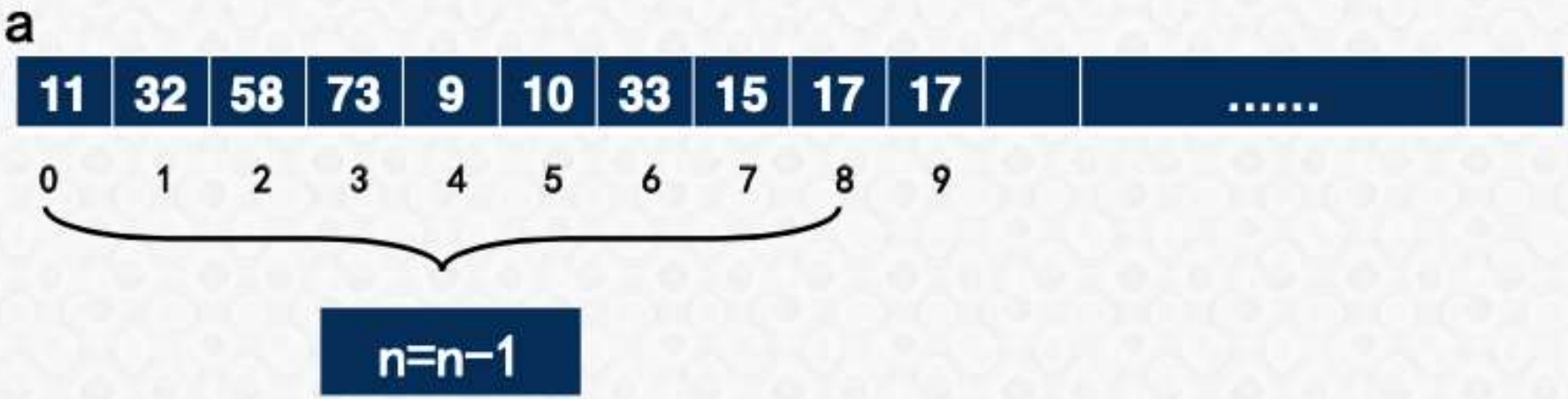



```
for(j=i;j<n-1;j++)
    a[j]=a[j+1];
```

注意：数组的最后一个元素实际上有两份拷贝

数组常用算法：删除

- 第三步，有效元素个数减一



数组常用算法：删除

- 例6_11 从整型数组a中删除第一个等于x的元素，如果x不是数组中的元素，则显示：“can not delete x!”。
- 算法：数组空间中的数据只能修改，不能“擦除”。
  - 确定待删除元素的位置
  - 元素从删除位置开始依次前移，覆盖待删除元素
  - 有效元素个数减一。
  - 注意：数组的最后一个元素实际上有两份拷贝
- 数据结构：

  数组a、变量x用于输入要删除的数据

  循环控制变量i，两次循环，查找位置以及移动

删除函数的实现

- 接口确定：
  - 功能
  - 入口：数组名、数组长度、待删除的数
  - 返回

```
#define SIZE 5
int main()
 {.....
    //初始化array、删除的x
    if(delArray(array,SIZE,x))
        print(array,SIZE-1);
    else
       printf("can not delete x!\n");
   ......
}
```

```
int delArray(int a[],int n,int x)
{
    int flag=1; /*标志位*/
   for (i=0;i<n &&a[i]!=x;i++) ;
                                /*查找x*/
   if (i==n)
            flag=0;
   else
   {
      for (j=i;j<n-1 ;j++)
          a[j]=a[j+1];  /*前移覆盖*/
   }
   return flag;
}
```

数组常用算法：排序

- 数组的概念
- 一维数组
- 二维数组
- 数组的常用算法
  - 查找
  - 插入
  - 删除
  - → 排序

数组常用算法：排序

- 通过元素位置的调整使得数组的所有元素按特定的顺序（升序、降序等）存放。
- 经典排序算法
  - 冒泡排序法（简单）
    - 向前冒泡
    - 向后冒泡

冒泡排序

- 例6.12：用冒泡法将n个元素按从小到大的顺序排序，然后输出排序后元素。

- 冒泡排序的算法思想：

> 在排序过程中对元素进行两两比较，越小的元素会经由交换慢慢“浮”到数组的前面（低下标处），像气泡一样慢慢浮起，由此得名。

冒泡排序

- 假设对长度为n的数组进行冒泡排序，算法可以描述如下：

➢ 第1趟冒泡：

从数组n-1下标的元素到0下标元素遍历，比较相邻元素对，如果后一个元素小于前一个元素，则交换。第一趟结束时，最小元素“浮”到0下标位置。

0 →	4	4	1
⋮	3	1	4
	1	3	3
n-1 →	5	5	5

冒泡排序

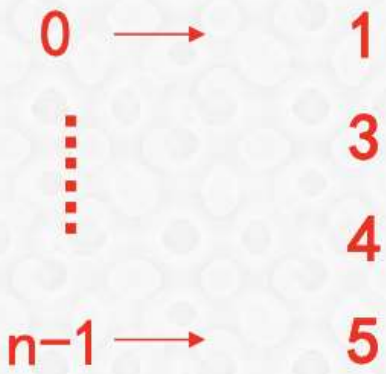
- 第2趟冒泡：

从数组n-1下标的元素到1下标元素遍历（因为0下标的已是最小元素，已经到位，无需再参加比较），比较相邻元素对，如果后一个元素小于前一个元素，则交换。第二趟结束时，本趟最小元素到达1下标位置。

0 →	1	1
⋮	4	3
	3	4
n-1 →	5	5

冒泡排序

- 第3趟冒泡：



- 依此类推，最多n-1趟冒泡（n是元素个数），便可以完成排序。

冒泡排序函数的实现

- 接口确定：
  - 功能：小到大排序
  - 入口：数组名、数组长度
  - 返回

```
#define SIZE 7
int main()
 {.....
    //输入array的长度和值
    BubbleSort(array,n);
     .......
}
```

```
void BubbleSort(int a[],int n)
{
   int temp;
   for (i = 0; i < n-1; i++)
                    /*共进行n-1趟排序*/
      for (j =n-1; j>i ; j--)
                    /*递减循环，从后往前比较*/
         if (a[j ] < a[j-1])
         {
            temp=a[j-1];
            a[j-1]=a[j];
            a[j]=temp;
         }
}
```

数组常用算法：排序

小结

- n个元素需要排序，要经过n-1趟冒泡就可以完成
- 每一趟排序中，是将未排序的元素进行两两比较
- 未排序的元素个数=元素总数-已排序的趟数

本章小结

- 一维数组的定义、初始化、元素的存储及访问
- 二维数组的定义、初始化、元素的存储及访问
- 一维数组中的经典算法：查找、插入、删除、排序等
- 二维数组的典型应用 ：矩阵、图形打印
- 一维、二维数组作为形参和实参的基本使用方法

输入理想的程序

输出快乐的人生