



Xidian University

C语言程序设计

Lec 5 数组





引言





简单问题：求4个整数的最大值

- 如何表示4个整数？ \Rightarrow 用4个整数变量
- 如何求出最大值？ \Rightarrow 逐一比较这4个变量

```
int main(){  
    int a1=75,a2=78,a3=91, a4=80;  
    int max=a1;  
    if( a2>max ) max=a2;  
    if( a3>max ) max=a3;  
    if( a4>max ) max=a4;  
    printf("max=%d\n", max);  
    return 0;  
}
```



引申：求 n ($n > 100$) 个整数的最大值

- 如果仍然采用前面的方法存在什么问题？

```
int main(){  
    int a1=80,a2=75,...,an=88;  
    int max=a1;  
    if( a2>max ) max=a2;  
    if( a3>max ) max=a3;  
    ...  
    if( an>max ) max=an;  
    printf("max=%d\n", max);  
    return 0;  
}
```

多个整数变量
表示繁琐

用于比较的
代码冗长



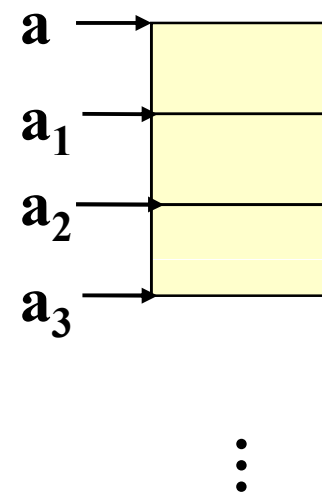
解决前面问题的思路：

- ✿ 将n个同类型变量以整体的形式表示
- ✿ 能够以简单的方式访问整体中的每个元素
- ✿ 已具备的知识
 - ✿ 每个变量在内存中都有对应的地址，知道该地址就可以访问变量： `int a=3; &a`
 - ✿ 每个变量占据的内存大小只取决于变量类型



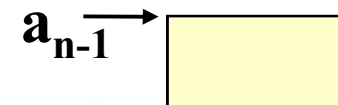
一种可能的解决方式

- ✱ 假设 n 个同类型变量在内存中连续存放，依次编号为 $0, 1, 2, \dots, n-1$
- ✱ 已知第一个变量的内存地址为 a
- ✱ 每个变量占据的内存大小为 $SIZE$



任意一个编号为 i 的变量对应的地址 a_i 可以通过以下公式计算：

$$a_i = a + i \times SIZE$$





需要解决的问题

- ❖ 如何才能让一组同类型变量连续存放？
- ❖ 第一个变量的地址如何得到？
- ❖ 能否直接通过编号引用任意一个变量，而将变量地址计算及通过地址访问变量的操作隐含在编号中？

解决以上问题需要引入一个新的概念
——数组



主要内容

- ✿ 数组定义和使用
- ✿ 二维和多维数组
- ✿ 字符数组和字符串





5.1 数组定义和使用





定义数组的方法

❖ 语法：元素类型 数组名称 [元素个数] ;

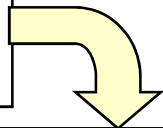
❖ 元素类型可以是任意类型

❖ 数组名称（数组变量）必须是合法标识符

❖ 数组元素个数必须是常量表达式^[注]

```
// 包含 100 个实数元素的数组  
float score[100];  
// 包含 20个字符元素的数组  
char name[20];
```

```
int n=100;  
float a[n];
```



在C99标准中允许用变量作为数组元素个数，但不支持C99的编译器上出现编译错误。VC6.0不支持



数组在内存中的实现

- 数组名是一个内存地址(不可修改)，称为数组首地址
- 数组元素从首地址开始连续存放

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

地址	数组元素	
0x1000	?	第1个元素
0x1004	?	第2个元素
0x1008	?	第3个元素
0x100C	?	第4个元素
0x1010	?	第5个元素

取数组首地址的三种方法

- 取数组名对应的值 `a`
- 取数组名的地址 `&a`
- 取第一个元素的地址 `&a[0]`

数组不能整体赋值

```
int a[8],b[8];  
a=b; //错误
```



访问数组元素

访问数组元素的方法：数组名[下标]

- 数组下标从0开始计数，最后一个元素下标是数组大小-1： $a[0]$ $a[10]$ $a[n-1]$ $a[5]=4$;
- 下标可以是常数、变量和表达式，但计算结果必须是整数： $a[3]$ $a[i]$ $a[i*2]=5$ 错误： $a[2*3.3]$

每个数组元素都是一个变量，可以赋值或取值

通过数组名和下标引用数组元素的本质是：

$\&a[i] = a + i \times \text{sizeof}(a[0])$

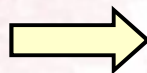
```
int a[5],i;
a[4]=0;
for(i=0;i<4;i++){
    a[i] = i;
    a[4] += a[i];
}
```



C语言数组的特性带来的问题

- ✚ C语言数组通过下标来计算元素的地址，
但**并未限制下标不能超出元素个数**

```
int a[4],i;  
a[4]=4;  
for( i=0; i<8; i++)  
    a[i] = i;
```



当数组下标大于等于数组元素个数时，编译不会出错，但**运行时**通常会出现内存访问错误

Note: 数组越界是程序中容易出Bug的地方，一定要注意



数组初始化

- ❖ 数组元素和变量一样应该先**初始化**才能使用
- ❖ **方法1**：先定义数组，然后用循环初始化每个元素

```
int a[4],i;  
  
for( i=0; i<4; i++)  
    a[i] = i;
```




数组初始化

❖ **方法2**：在定义数组时指定元素的初始值

❖ **格式**：所有初值用**大括号**包围，**逗号**分隔

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

a[0]=0 a[1]=1
a[2]=2 a[3]=3

数组大小和初值
个数相同时，一
一对应初始化

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

a[0]=0 a[1]=1
a[2]=2 a[3]=0
a[4]=0

数组大小大于初
值个数时，其余
元素用0初始化

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

a[0]=0 a[1]=1
a[2]=2 a[3]=3

不指定数组大小
时，数组大小和
初值个数相同

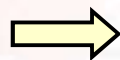


数组初始化

❖ 初始化数组时易犯的**错误**

- ❖ 使用**方法1**时，数组**下标越界**

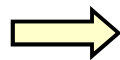
```
int a[4],i;  
  
for( i=0; i<8; i++)  
    a [i] = i;
```



运行时错误

- ❖ 使用**方法2**时初值个数大于数组元素个数

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



编译错误



例1：求100个整数的最大值

//不使用数组的程序

```
int main(){  
    int a1=80,a2=75,...,a100=88;  
    int max=a1;  
    if( a2>max ) max=a2;  
    if( a3>max ) max=a3;  
    ...  
    if( a100>max ) max=a100;  
    printf("max=%d\n", max);  
    return 0;  
}
```

表示多
个整数

用于比较
的代码

//使用数组的程序

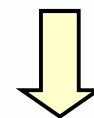
```
int main(){  
    int a[100]={/*初值*/};  
    int max=a[0],i;  
    for(i=1;i<100;i++)  
        if( a[i]>max ) max=a[i];  
    printf("max=%d\n", max);  
    return 0;  
}
```



例2：计算Fibonacci数列的前40项

$$F(n) = \begin{cases} 1 & n=0 \\ 1 & n=1 \\ F(n-1)+F(n-2) & n>1 \end{cases} \Rightarrow \begin{aligned} &F(0)=1 \\ &F(1)=1 \\ &F(n)=F(n-1)+F(n-2) \quad n>1 \end{aligned}$$

如果把圆括号换成方括号？



这其实就是
数组的表示
方式

$$F[0]=1$$

$$F[1]=1$$

$$F[n]=F[n-1]+F[n-2] \quad n>1$$



例2：计算Fibonacci数列的前40项

```
int main(){
    int F[40]={1,1}; // F(0)=1, F(1)=1
    int i;
    for(i=0; i<40; i++){
        if( i>1 ){ // i>1时, F(i)=F(i-1)+F(i-2)
            F[i] = F[i-1] + F[i-2];
        }
        printf("F (%d)=%d\n", i, F[i] ); //输出F(i)
    }
    return 0;
}
```



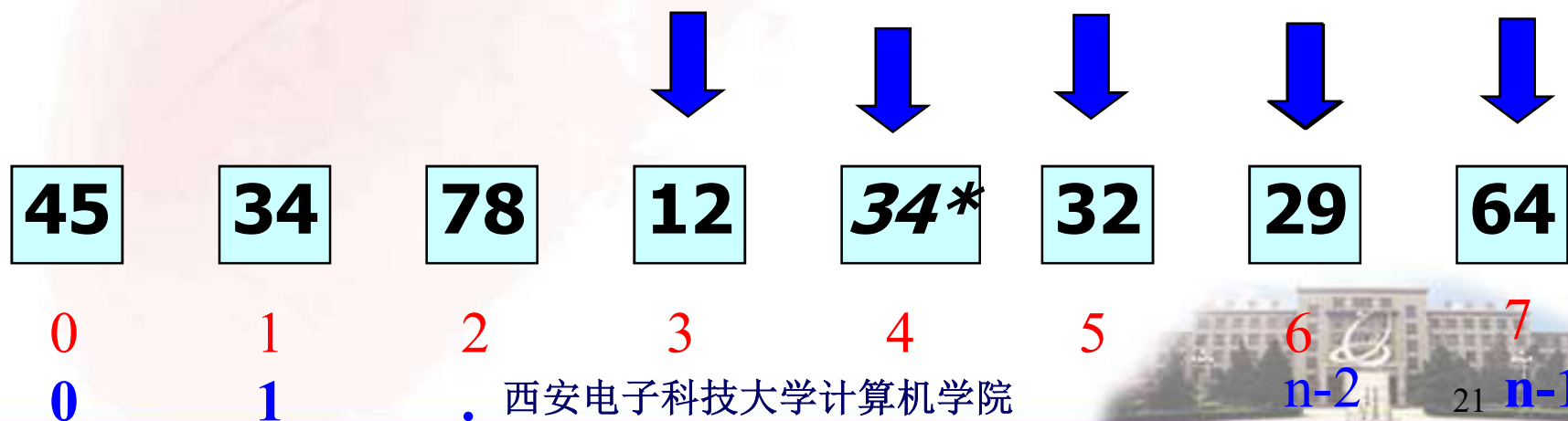
例3：输入100个学生的成绩，按从低到高排序后输出

- ❖ 100个学生成绩可以用一个具有100个元素的数组表示
- ❖ 排序的方法？
 - ❑ 选择排序、起泡排序
 - ❑ 快速排序、归并排序、堆排序





❁ 第一趟在 n 个元素中选取最小元素作为有序序列的第一个元素，第二趟在 $n-1$ 个元素中选取最小元素作为有序序列的第二个元素，第 i 趟在 $n-i+1$ 个元素中选取最小的元素作为有序序列中的第 i 个元素。



//对数组进行选择排序

```
int a[100];
```

```
int n=100; //数据量
```

```
int i=0,k=0; //循环变量
```

```
int t=0; //用来交换的临时变量
```

```
int min=0; //用来记录最小值
```

```
for(i = 0; i < n-1; i++) { //i<=n-2; 也行
```

```
    for(k = i+1, min = i; k < n; k++)
```

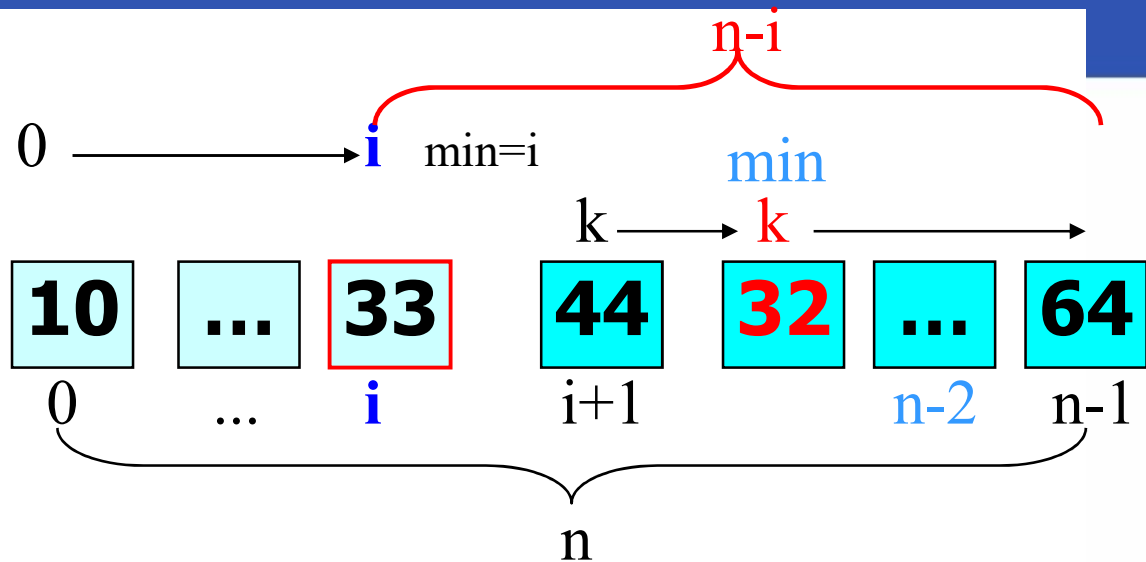
```
        if (a[k]<a[min]) min = k;
```

```
    if (min != i) {
```

```
        t=a[i];a[i]=a[min];a[min]=t;
```

```
    }
```

```
}
```



找 $n-i$ 个元素中最小值对应的下标 j

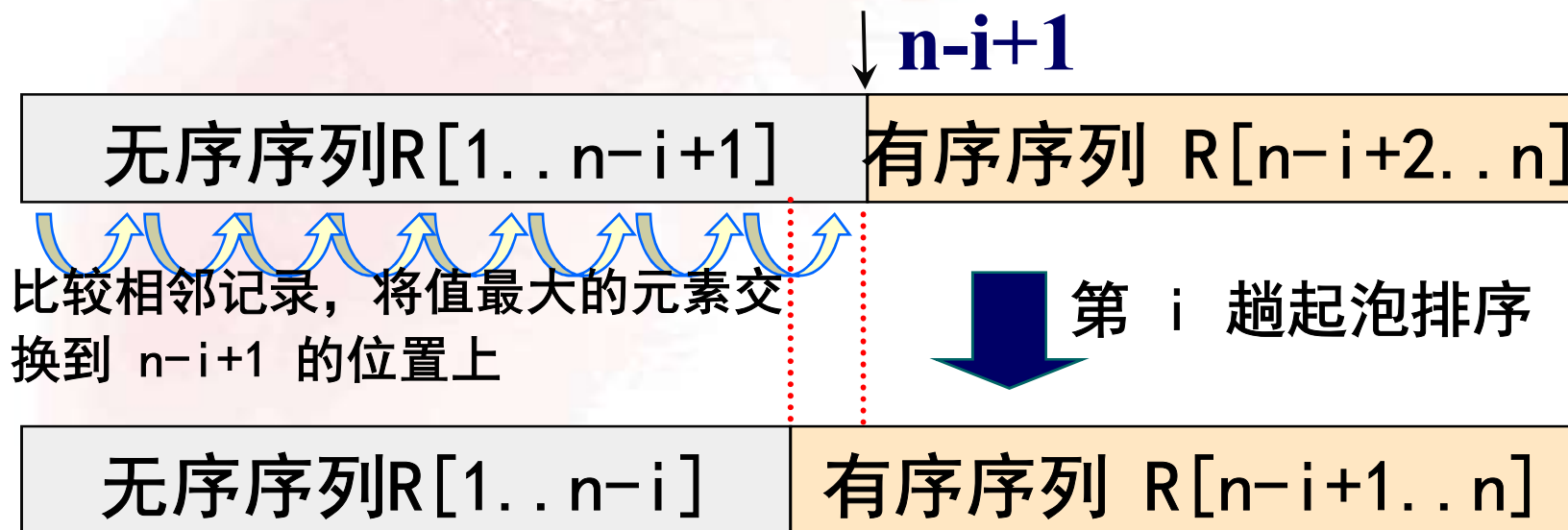
如果最小值不是 i 元素, 则将 $a[i]$ 和 $a[j]$ 交换

demo_5_sort.c



(2) 起泡排序算法

- ❁ 起泡排序 (Bubble Sorting) 的基本思想是：将相邻位置的元素进行比较，若为逆序则交换之。若在一趟排序过程中没有进行过交换记录的操作，则整个排序过程终止。





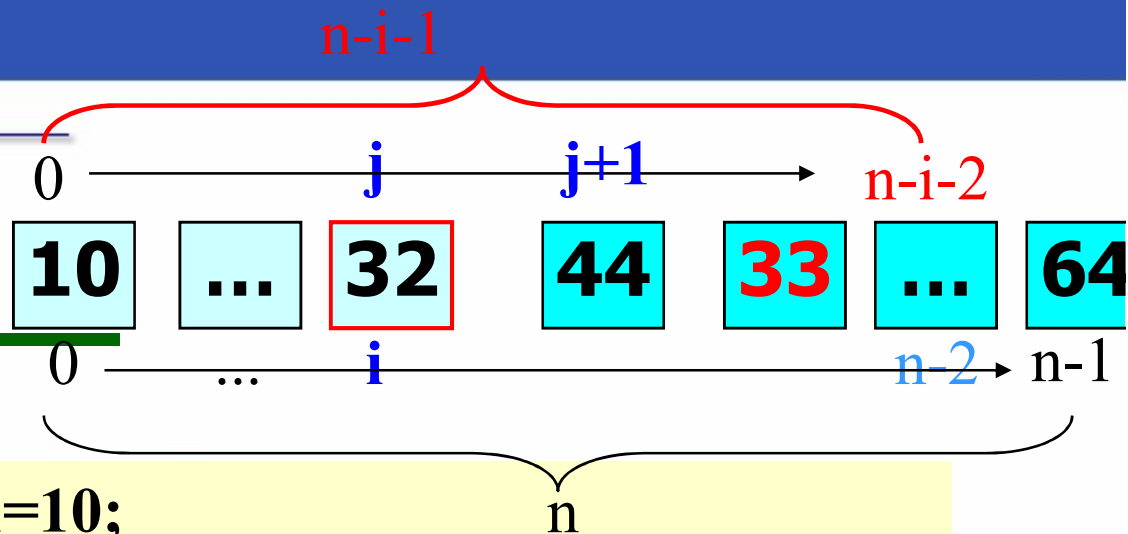
起泡排序过程示例

	0	1	2	3	4	5	6	7
初始数组:	49	38	65	97	76	13	27	49*
第一趟排序后:	38	49	65	76	13	27	49*	97
第二趟排序后:	38	49	65	13	27	49*	76	97
第三趟排序后:	38	49	13	27	49*	65	76	97
第四趟排序后:	38	13	27	49	49*	65	76	97
第五趟排序后:	13	27	38	49	49*	65	76	97
第六趟排序后:	13	27	38	49	49*	65	76	97

n个数，
最不理想
情况下需
要翻:n-1
轮



起泡排序算法



```
int a[10], i, j, change, t, n=10;
```

```
for( i = 0, change =1; i <=n-2 && change; ++i){ // 也可写成i<n-1
```

```
    change =0;
```

```
    for(j = 0; j <=n- i -2; ++j){ //写成j<n-i-1; 也可以
```

```
        if (a[j] > a[j+1] ) {
```

```
            t = a[j];  a[j] = a[j+1]; a[j+1] = t;
```

```
            change =1;
```

```
        }
```

```
    }
```

```
}
```



排序算法应用

❖ **例1**：写一个程序，对给定的整数序列排序，然后求排序后序列的**中位数**。（一个有序序列的中位数是序列**中间位置上的数**，如果该序列长度为**偶数**，则中位数取中间**两个数的平均值向下取整**）。

步骤：

1. 输入数据保存到一维数组
2. 对数组排序
3. 求中位数



数组搜索

✿ 搜索一个数组以确定一个特定数值的位置

- (1) 假定目标还没找到。
- (2) 从首个数组元素开始。
- (3) 在目标还未找到，并且还有数组元素未搜索到时，
重复进行以下操作

如果目前的元素与目标匹配

设置一个标志，结束搜索。

否则

前进到下一个数组元素。



数组搜索

```
int main(){
    int a[10]={ /*数组元素初值...*/ };
    int target=50;
    int found=0; //搜索标志
    int i; //循环变量
    for(i=0; i<10 && !found; i++){
        if( a[i]==target ) found=1;
    }
    if(found)
        printf("target found, pos=%d\n", i-1 );
    else
        printf("target not found\n");
}
```



数组搜索

✿ 折半（二分）搜索

■ 在有序数组上的快速搜索方法

- (1) 设 **low** 是第一个数组元素的下标，**high** 是最后一个数组元素的下标，**found** 为假。
- (2) 只要 **low** 不大于 **high**，且目标值尚未找到，就重复下列步骤。
 - (3) 设 **mid** 是在 **low** 和 **high** 之间的中间元素的下标。
 - (4) 如果 **mid** 处的元素是目标值。
将 **found** 设为真，**index** 设为 **middle**。
 - 否则如果 **mid** 处的元素大于目标值。
将 **high** 设为 **mid - 1**。
 - 否则
将 **low** 设为 **mid + 1**。





$$O(n/2)$$
$$O(\log_2 n)$$

mid=(low+high)/2





折半（二分）搜索：搜索成功的例子

target=23, low=0, high=7, mid=3

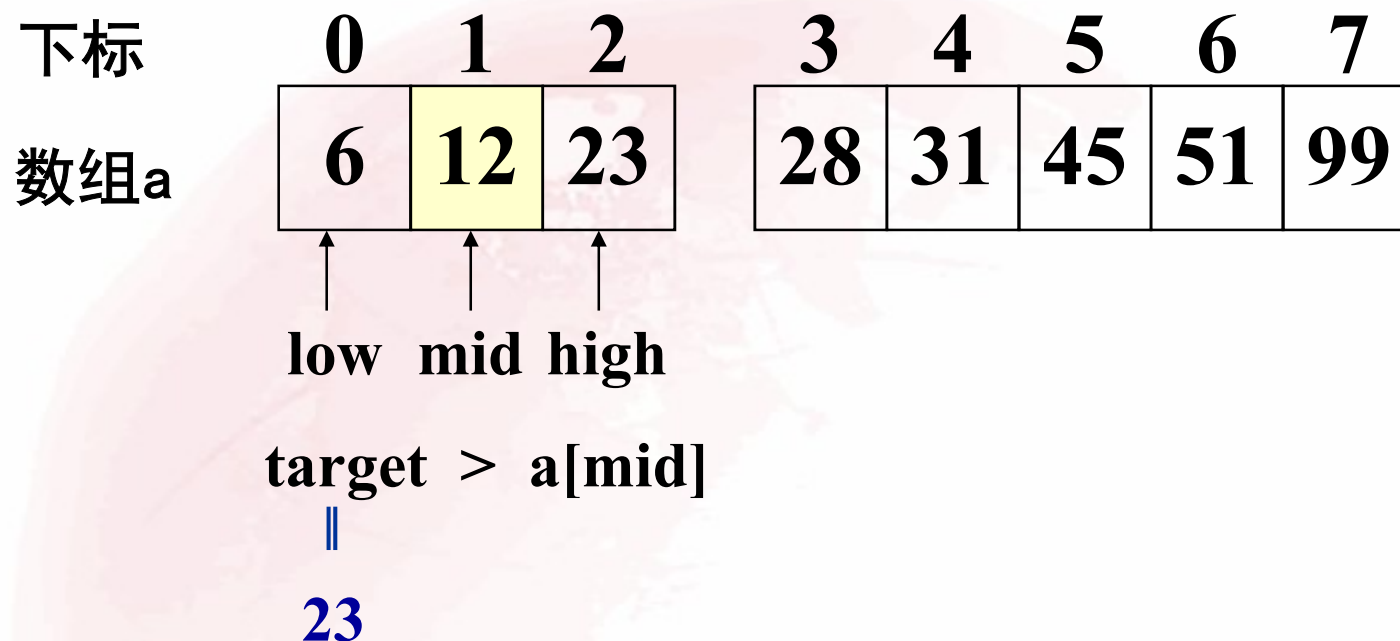
下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99
	↑			↑				↑
	low			mid				high

target < a[mid]
||
23



折半（二分）搜索：搜索成功的例子

target=23, low=0, high=2, mid=1





折半（二分）搜索：搜索成功的例子

target=23, low=2, high=2, mid=2

下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99

low high

mid

target 等于 a[mid]，结束搜索，搜索成功

||
23



折半（二分）搜索：搜索失败的例子

target=55, low=0, high=7, mid=3

下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99
	↑			↑				↑
	low			mid				high

target > a[mid]

||
55



折半（二分）搜索：搜索失败的例子

target=55, low=4, high=7, mid=5

下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99

low mid high

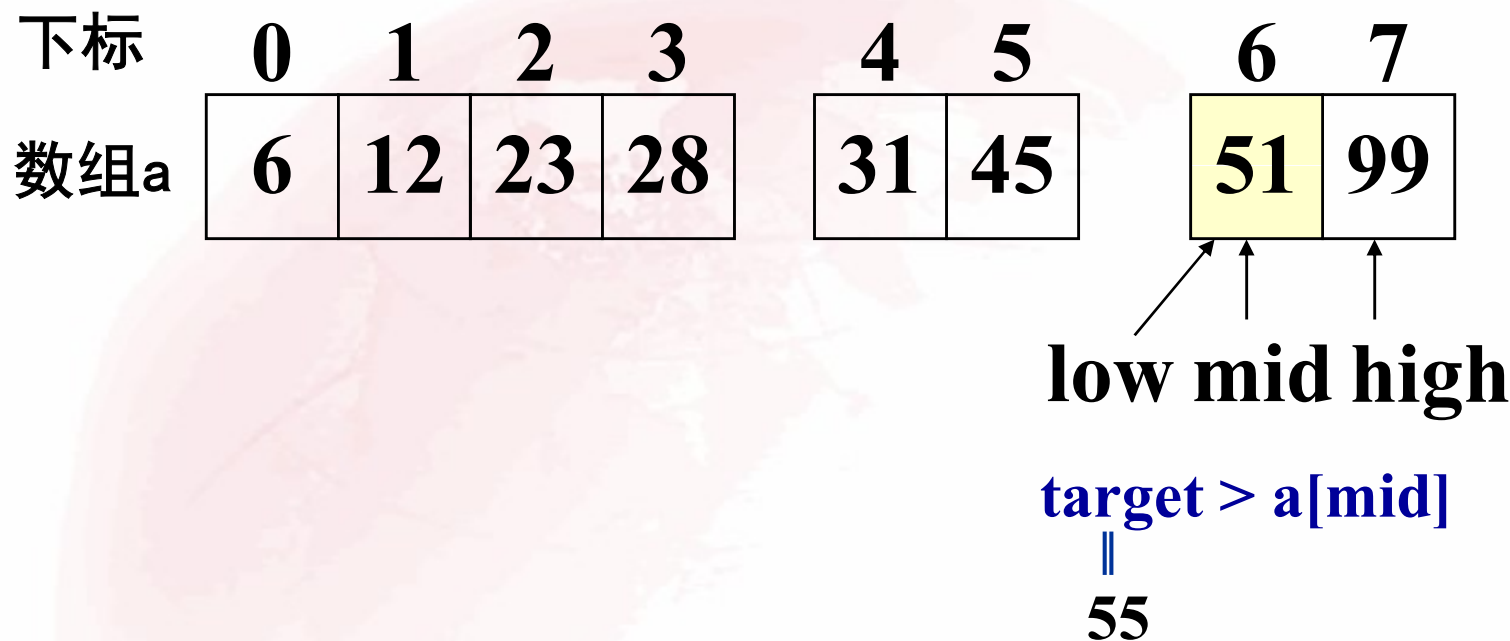
target > a[mid]

55



折半（二分）搜索：搜索失败的例子

target=55, low=6, high=7, mid=6





折半（二分）搜索：搜索失败的例子

target=55, low=7, high=7, mid=7

下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99

low high
mid

target < a[mid]
||
55



折半（二分）搜索：搜索失败的例子

target=55, low=7, high=6

下标	0	1	2	3	4	5	6	7
数组a	6	12	23	28	31	45	51	99

high low

low>high, 结束查找, 查找失败

demo_5_half_search.c

```
int main() {  
    int a[10]={ /*数组元素初值...*/ };  
    int target=50;  
    int found=0; //搜索标志  
    int low=0,mid; high=n-1;  
    while ( low<=high&&!found) {  
        mid=(low+high)/2;  
        if (a[mid]==target) found=1;    //查找成功  
        else if (target<a[mid]) high=mid-1; //下一次到前半区查找  
        else low=mid+1;                //下一次到后半区查找  
    }  
    if(found)    printf("target found, pos=%d\n", mid );  
    else        printf("target not found\n");  
    return 0;  
}
```



练习

- 编写程序将数组x中的整数以相反顺序复制到数组y中（也就是y[0]中保存x[n-1]，...，y[n-1]中保存x[0]）

x 12, 15, 67, 23, 55, 44



y 44, 55, 23, 67, 15, 12



5.2 二维和 multidimensional 数组

- 一维数组处理线性结构

- 二维数组可以处理矩阵

- 例如一幅 200×300 的图像
- 一个行列式

- C语言对二维数组的表示

- 内存是一维线性编址，没有行列的概念
- C语言将二维数组的一行作为一个一维数组，二维数组就是连续多个一维数组
- 多维数组以此类推

200

300



128

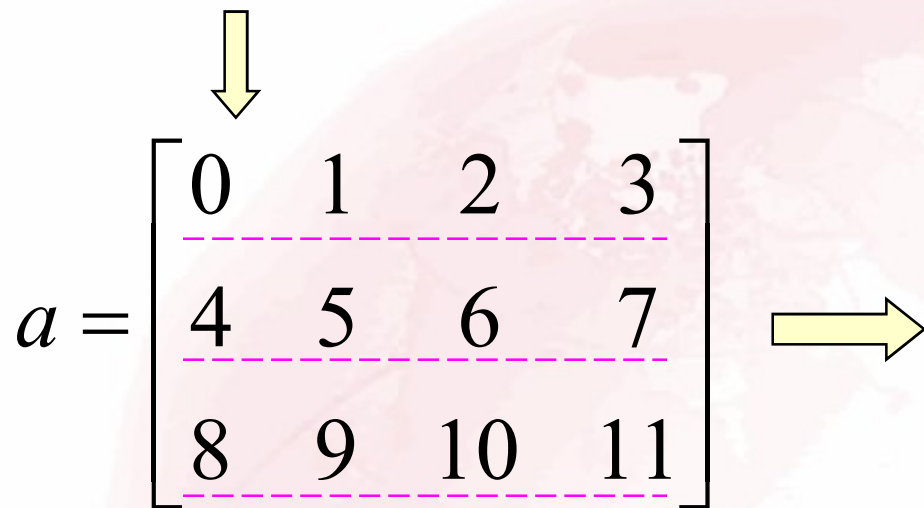
$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$



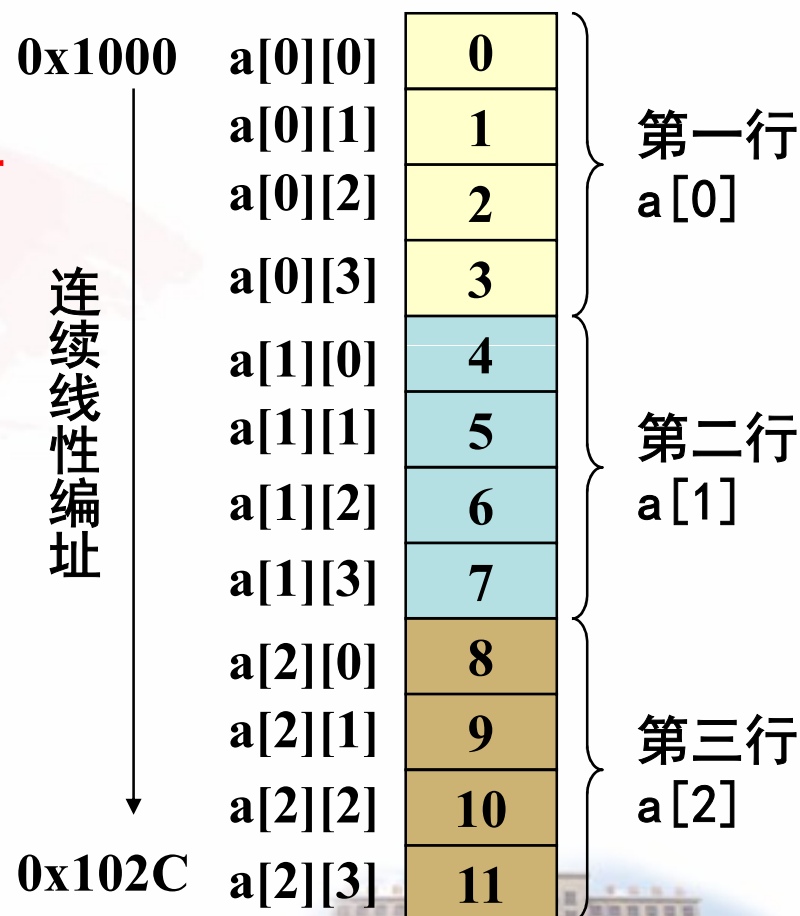
C语言对二维数组的表示

`int a[4];` //一个有4元素的一维数组

`int a[3][4];` //一个三行四列的二维数组



也可以认为a是一个3个元素的一维数组，每个元素都是有4个元素的一维数组





二维数组初始化

✿ 方法1：按行初始化

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

↓

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

每一对大括号内是一行的初值，
初值个数不足时用0初始化

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

↓

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

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

↓

$$a = \begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & 0 \end{bmatrix}$$



二维数组初始化

❖ 方法2：将二维数组当作一维数组初始化

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



$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

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



$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 0 & 0 \end{bmatrix}$$

大括号内是整个二维数组的初值，按照**行序**依次赋值，初值个数不足时用0初始化



二维数组初始化

❖ 方法3：省略第一维大小

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

↓

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

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

↓

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 0 \end{bmatrix}$$

定义二维数组时，如果有初值，**可以省略第一维大小**（行数）**不能省略第二维大小**（列数），行数根据初始值个数计算

定义二维数组时，如果**没有初值**，第一维大小（行数）和第二维大小（列数）**都不能省略**



二维数组的使用

- ✱ 用行下标和列下标引用单个元素，行列下标都从0开始计数

✱ $a[i][j]$ 表示二维数组 a 的 i 行 j 列的元素

- ✱ 示例1：求一个方阵的主对角线之和

demo_5_multi_array.c

```
int a[3][3]={...};  
int sum=0,i;  
for( i=0; i < 3; i++){  
    sum += a[i][i];  
}
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

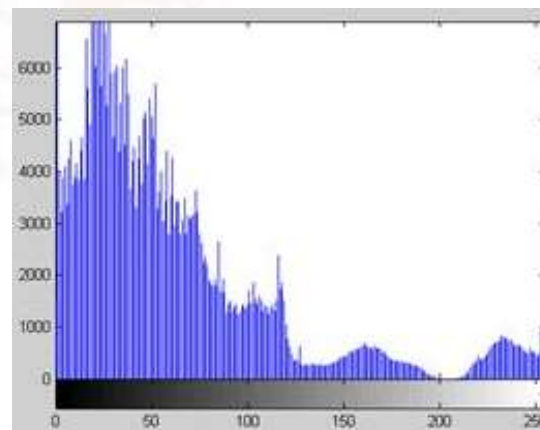
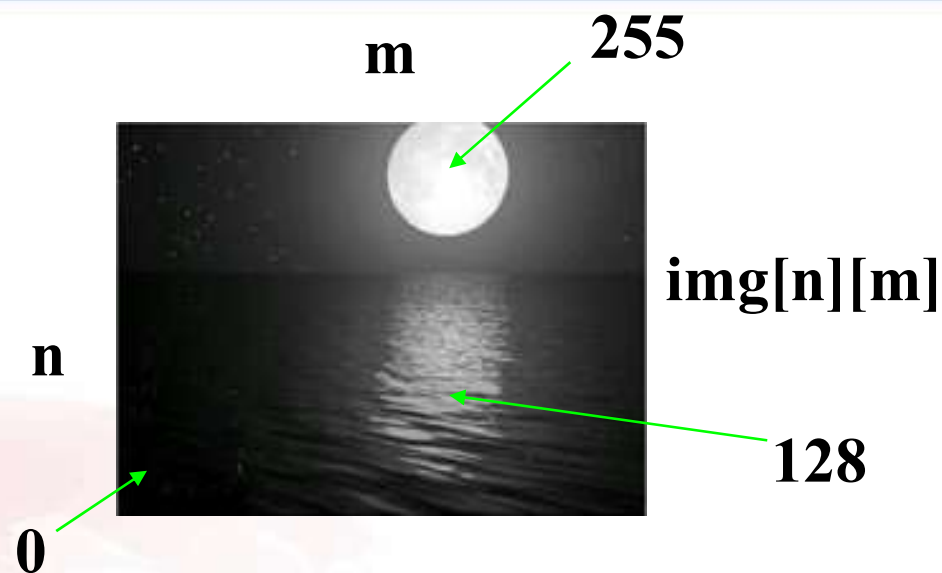


二维数组的使用

❁ 示例：图像的灰度直方图

一幅 $n \times m$ 的灰度图像可以用一个二维矩阵表示，矩阵中的每个元素表示对应像素的灰度值。

“灰度直方图”以图像中每种灰度级的像素个数来反映图像中每种灰度出现的频率



$a[256]$

$a[i]++$

histogram
(直方图)



二维数组的使用

❖ 示例：图像的灰度直方图

一幅 $n \times m$ 的灰度图像可以用一个二维矩阵表示，矩阵中的每个元素表示对应像素的灰度值。

“灰度直方图”以图像中每种灰度级的像素个数来反映图像中每种灰度出现的频率

```
int main(){
    int img[256][256],a[256]={0};
    int m,n,k,i,j;

    scanf("%d %d %d",&m,&n,&k);
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            scanf("%d",&img[i][j]);
            a[img[i][j]]++;
        }
    }
    for(i=0;i<k;i++){
        printf("%d %d\n",i,a[i]);
    }
    return 0;
}
```



二维数组的使用

- ❁ 示例3：请写一个程序，找出给定矩阵的马鞍点。
若一个矩阵中的某元素在其所在行最小而在其所在列最大，则该元素为矩阵的一个马鞍点，
如果找到输出对应的行和列，找不到输出“no”

11	13	121
407	72	88
23	58	1
134	30	62

马鞍点，行为1，列为1

思路：先找出每行最小值及其列下标，然后检查该值是不是对应列上的最大值，是则输出行列坐标，否则看下一行，直到所有行检查完



示例3伪代码

demo_5_multi_maan.c

```
for (row=0; row<行数; row++) { //外层循环处理每一行
    找到一行的最小值对应的列下标col, 可以用循环完成
    当前行中值最小的元素是a[row][col]
    for (i=0; i<行数; i++) { //内层循环处理一列
        判断a[row][col]是不是这一列a[i][col]中最大的值
    }
    if (是马鞍点) {
        输出行列坐标
        马鞍点个数递增
    }
}
if (马鞍点个数==0) 输出"no"
```



5.3 字符数组和字符串

- ✿ 文字处理是应用程序中一个常用的功能
- ✿ **字符数组**就是元素类型是字符的数组
 - ▣ 定义方式和普通数组一样
 - ▣ 使用方式和初始化方式也一样

```
char name[20]={'L', 'g', 'x'};
```



字符数组name的其余元素用**数值0**初始化，不是字符'0'，这个0对字符串而言有**特殊意义**

字符串结束符: `\0`, 数值 0

```
char name[10]={'L', 'g', 'x'};
```

“Lgx”

字符串结束符



L	g	x	\0	\0	\0	\0	\0	\0	\0
---	---	---	----	----	----	----	----	----	----

76	103	120	0	0	0	0	0	0	0
----	-----	-----	---	---	---	---	---	---	---

```
char name[10]={'c', 'h', 'i', 'n', 'a'};
```

“china”

字符串结束符



c	h	i	n	a	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----

99	104	105	110	97	0	0	0	0	0
----	-----	-----	-----	----	---	---	---	---	---

```
char name[10]={'L', 'g', 'x', '0'};
```

“Lgx0”

字符串结束符



L	g	x	0	\0	\0	\0	\0	\0	\0
---	---	---	---	----	----	----	----	----	----

76	103	120	48	0	0	0	0	0	0
----	-----	-----	----	---	---	---	---	---	---

char name[10]={‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’};

“123456789”

49	50	51	52	53	54	55	56	57	0
----	----	----	----	----	----	----	----	----	---

“1234567891”

49	50	51	52	53	54	55	56	57	49
----	----	----	----	----	----	----	----	----	----

“1234567890”

49	50	51	52	53	54	55	56	57	48
----	----	----	----	----	----	----	----	----	----

“123”

49	50	51	0	53	54	55	56	57	48
----	----	----	---	----	----	----	----	----	----

‘\0’,



从键盘输入一行字符，保存到字符数组中

```
#include <stdio.h>
int main(){
    char line[80]={0}; //如果不初始化为0会导致什么结果?
    int i;
    for(i=0;i<80;i++){
        scanf("%c",&line[i]); //输入一个字符
        if(line[i]=='\n')break; //输入字符是换行符时结束输入
    }
    printf("%s",line);
    return 0;
}
```




字符串的本质

❁ 字符串是一个字符数组加结束标志（'\0'）

- ❁ 定义字符串就是定义一个字符数组
- ❁ 字符串用字符数组存储
- ❁ 字符串存储空间比实际字符个数多1
- ❁ 结束标志是为了知道字符串什么时候结束
- ❁ 字符串长度是字符数组中第一个结束标志前的字符个数
 - 字符串长度可能小于字符数组大小
 - 字符数组中没有结束标志时，长度不确定



字符串初始化

❁ 方法1: 像数组一样单独指定每个元素

```
char name[20]={'C','h','i','n','a'};
```

数组大小为20，字符串长度为5，其余字符初值为'\0'

```
char name[]={'C','h','i','\0','a'};
```

数组大小为5，字符串长度为3

```
char name[5]={'C','h','i','n','a'};
```

数组大小为5，缺少结束标志'\0'，不能当作字符串使用

```
char name[]={'C','h','i','n','a'};
```

数组大小为5，缺少结束标志'\0'，不能当作字符串使用



字符串初始化

❁ 方法2：用双引号括起来的字符串常量初始化

```
char name[20] = "China";
```

数组大小为20，字符串长度为5，其余字符初值为'\0'

```
char name[] = "China";
```

数组大小为6，字符串长度为5，自动添加结束标志'\0'

```
char name[5] = "China";
```

数组大小为5，没有结束标志'\0'，不能当作字符串使用



字符串使用示例

- ❁ 示例1：写一个程序，将一个字符串复制到一个字符数组里（同样做成字符串）。

```
int main() {  
    char dst[100], src[]="hello,world";  
    int i = 0;  
    while ( src[i] != '\0' ) {  
        dst[i] = src[i];  
        ++i;  
    }  
    dst[i] = '\0'; //添加结束标志  
    return 0;  
}
```

将字符串src中的字符复制到字符数组dst中，形成一个字符串，要求dst足以容纳src所有字符和结束标志



字符串使用示例

❁ 示例1的其他实现方式。

```
int main () {  
    char dst[100], src[]="hello,world";  
    int i = 0;  
    while ( (dst[i] = src[i]) != '\0' ) ++i;  
    return 0;  
}
```

```
int main () {  
    char dst[100], src[] = "hello,world";  
    int i = 0;  
    while ( dst[i] = src[i] ) ++i;  
    return 0;  
}
```

这两个简化版本利用
赋值表达式的值作为
循环结束条件

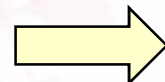




字符串使用示例

- ❁ 示例2：写一个程序将字符串转换为整数，例如将字符串 “12345”转换为整数12345

$$\begin{aligned} & d_n d_{n-1} \dots d_1 d_0 \\ & \parallel \\ & d_n d_{n-1} \dots d_1 \times 10 + d_0 \\ & \parallel \\ & ((d_n d_{n-1} \dots d_2 \times 10) + d_1) \times 10 + d_0 \\ & \parallel \\ & \dots \end{aligned}$$



```
int main(){
    int n=0,base=10,i;
    char s[]="12345";
    for( i=0; s[i]!='\0'; i++ ){
        n=n*base+(s[i]-'0');
    }
    printf("n=%d\n",n);
    return 0;
}
```

如果要转换其他进制的数，只需将base改为对应的进制即可



Xidian University



西安电子科技大学计算机学院

