

C语言与程序设计

The C Programming Language



第7章 数组

华中科技大学计算机学院

黄宏



第7章 数组

用来描述一群有联系的同类型数据集合

- ◆ 一维数组的声明、初始化和使用
- ◆ 数组作为函数参数的使用
- ◆ 字符串数组
- ◆ 多维数组

7.1 数组概述

用来描述一群有联系的同类型数据集合

```
#define SIZE 30  
int score[SIZE];
```

score是含有30个元素的int型数组

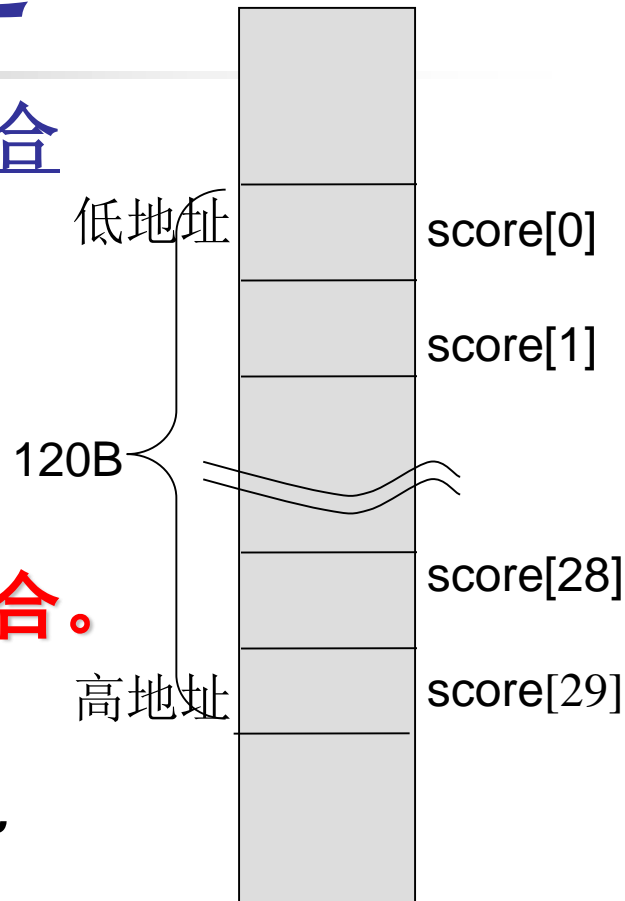
数组：是固定数量的同类型元素的集合。

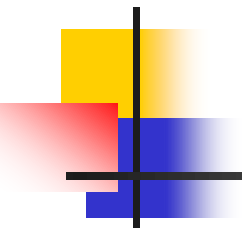
➤ 数组元素的下标从0开始

数组score的30个元素是：score[0] ~
score[29]

➤ 各个元素在内存中连续存放

数组score所占内存：sizeof (int) * SIZE





7.2 一维数组

- 只有一个下标
- 可用于表示一个线性的数据队列



7.2.1 一维数组的声明

存储类型 **类型说明符** **数组名[常量]={初值表} ;**

- 蓝色部分可选
- 不能是 register
- `int score[SIZE];`
- `static int y[10];`
- `extern double s[2];`

7.2.2 一维数组元素的使用

数组名[下标表达式]



整型表达式

整型常量

整型变量

含有运算符的整型表达式

值为整型的函数调用

$a[2]$ 、 $a[i]$ 、 $a[i+j]$ 、 $a[\max(a,b)]$

代表了一个元素， 等价一个同类型的变量。

下标值不要超过数组的范围

```
int a[4],i;
```

```
for(i=0;i<=4;i++)
```

```
    a[i]=i+1;
```

$a[4]=5;$

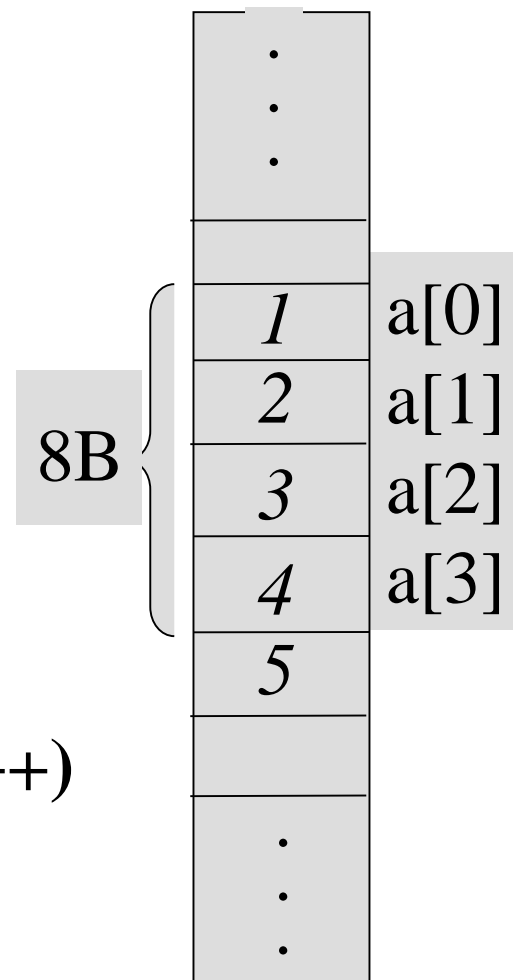
↑
下标越界

**破坏数组以外的其它变量的值,
可能造成严重后果**

```
int a[4],i;
```

```
for(i=0;i<4;i++)
```

```
    a[i]=i+1;
```





7.2.3 一维数组的初始化

存储类型 类型 数组名[常量表达式]={初值};

- 对全部元素赋初值（初值的个数与数组长度相同）

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

<=> int x[]={1,2,3,4,5}; /*数组长度由初值个数确定*/

- 对部分元素赋初值(初值的个数小于数组长度)

int z[6]={1,2,3,4}; /* 前4个下标变量赋值 */

z[0]:1 z[1]:2 z[2]:3 z[3]:4



【例】使用一维数组计算学生的平均成绩

- 如何将计算n个学生的成绩定义成函数？

函数名： **CalAverage**

形式参数：

若干成绩： 数组 **score**

人数： **n**

返回值： 平均成绩

- 函数原型

double CalAverage(int score[], int n);

- 算法步骤：

- (1) 成绩的累加求和存放于**sum**;
- (2) 计算并返回平均成绩**average**;

```
double CalAverage(int x[ ], int n)
{
    int sum,i;
    for(sum=0,i=0;i<n;i++)    /* 累加求和 */
        sum+=x[i];
    return (1.0*sum/n);    /* 计算并返回平均值 */
}
```

程序

```
#include <stdio.h>
#define SIZE 30 /* 学生 */
double CalAverage(int x[ ], int );
int main(void)
{
    int score[SIZE], i, sum, n;
    double average;
    printf( "输入学生人数(<= %d): \n", SIZE);
    scanf( "%d", &n); /* 输入人数存于n */
    printf( "输入%d个学生的成绩: \n", n);
    for(i=0; i<n; i++) /* 输入成绩存于数组score中 */
        scanf( "%d", &score[i]);
    printf( "average= %f \n", CalAverage(score, n));
    return 0;
}
```

&score[0]

8.2.5 一维数组作为函数参数

■ 子函数

- 给出形参数组的声明和形参数组元素的个数（char数组除外）
- 形参是不指定大小的数组

double **CalAverage** (int score[], int n);

■ 调用函数

- 实参的值是**数组名**或数组元素的地址

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

double ave;

ave=CalAverage(x, 5);

ave=CalAverage(&x[1], 3);

数组的首地址 $x \longleftrightarrow \&x[0]$

一维数组作为函数参数的例子

```
#include <stdio.h>
```

```
void fun(int y[ ], int n);
```

```
void main(void)
```

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

```
    fun(x,5);
```

```
    for(i=0;i<=4;i++) printf( "%d" ,x[i]);
```

```
    fun(&x[2],3); printf( "\n" );
```

```
    for(i=0;i<=4;i++) printf( "%d" ,x[i]);
```

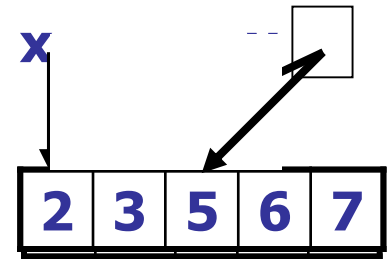
```
}
```

```
void fun(int y[ ], int n)
```

```
{ int i;
```

```
    for(i=0; i<n; i++)    y[i]++;
```

```
}
```





C的参数传递：传值和传址

用数组名作参数时，则为地址传送，即实参数组的首地址传递给形参数组首地址。所以，实参、形参数组共享相同的内存单元。相当于是值的双向传递



例 对 n 个整数采用冒泡法对其排序

排序问题——计算机处理数据的一个重要问题

➤ 将**冒泡排序算法**定义成函数

- 函数名: *BubbleSort*

- 形式参数:

 - 带排序的若干整数: 数组 *a*

 - 整数的个数: *n*

- 返回值: 无

- 函数原型

 - void BubbleSort(int a[], int n);*

- 算法?



冒泡 (bubble) 排序法

基本思路： 设有 n 个数，需将它们从小到大顺序排列。则：

- 1. 对 n 个元素，依次比较相邻的两个数，小的调到前头，比较 $(n-1)$ 次后，最大的一个数“沉底”。
- 2. 对剩下的 $n-1$ 个元素，比较 $(n-2)$ 次后，得到次大的数。
- 3. 对上述过程重复，直至剩下一个元素。

例: `int a[4] = {10 8 5 7}`

```
n个数进行 ? 轮比较
for(i=1;i<n;i++){
    相邻元素的一一比较, 交换
}
```

第1轮:

10	8	8	8
8	10	5	
5	5	10	7
7	7	7	10

第*i*轮中要进行 ? 次两两比较

第2轮:

8	5	5
5	8	7
7	7	8
10	10	10

```
for(i=1;i<n;i++){
    for(j=0;j<n-i;j++){
        if(a[j]>a[j+1])
            a[j]与a[j+1]交换
    }
}
```

第3轮:

5
7
8
10

比较1次 交换0次

```
#include<stdio.h>
```

```
#define N 10
```

```
void BubbleSort ( int a[ ],int n)
```

```
{
```

```
    int i, j, t;
```

```
    for (i=1; i<n; i++) /* 共进行n-1轮"冒泡" */
```

```
        for (j=0; j<n-i ; j++) /* 对两两相邻的元素进行比较 */
```

```
            if (a[j]>a[j+1] ) { t=a[j]; a[j]=a[j+1]; a[j+1]=t; }
```

```
    }
```

```
int main ( )
```

```
{ int x[N], i;
```

```
    printf (" please input %d numbers: \n ", N);
```

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

```
    BubbleSort (x , N ) ;
```

```
    printf (" the sorted numbers: \n ");
```

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

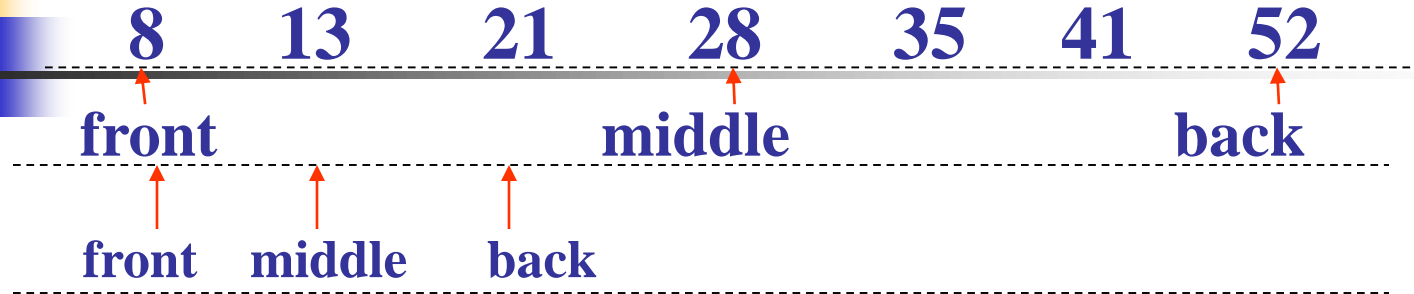
```
    return 0;
```

```
}
```

7.6.2 基于分治策略的二分查找

- 二分查找（也称折半查找）算法的思路
 - 将已排序的 n 个元素(存放于数组 a)分成两半，取 $a[n/2]$ 与 x 比较
 - 如果 $x=a[n/2]$ ，则找到 x ，算法结束
 - 如果 $x<a[n/2]$ ，则在数组 a 的前半部分继续查找 x
 - 如果 $x>a[n/2]$ ，则在数组 a 的后半部分继续查找 x
- 返回值
 - 如果找到 x ，返回 x 在数组 a 中的位置（下标）
 - 如果没有找到，返回-1

在 $n=7$ 个数中 找 $x=14$



1) 令 $front = 0$, $back = n - 1$

2) $middle = (front + back) / 2$

3) 将 x 与 $a[middle]$ 比较

◆ $x < a[middle]$, x 在 $a[0]$ 与 $a[middle-1]$ 范围内
令 $back = middle - 1$

◆ $x > a[middle]$, x 在 $a[middle+1]$ 与 $a[back]$ 范围内
令 $front = middle + 1$

◆ $x == a[middle]$, x 在数组 a 中的下标为 $middle$

转2, 直至 $top > bottom$

二分查找函数

/ find x in a[0]~a[n-1] (sort ascending)
return the subscript , or -1 if no match */*

```
int BinarySearch(int a[ ], int x, int n)
{ int front=0, back=n-1, middle;
  while(front<=back) {
    middle=(front+back)/2; /* 计算中间单元的下标 */
    if(x<a[middle]) back=middle-1; /* 查找单元变成原来的前半部*/
    else if(x>a[middle]) front=middle+1; /*在后半部查找*/
    else return (middle); /* 找到，返回下标 */
  }
  return -1; /* 没有找到，返回-1 */
}
```



二分查找函数

```
#include<stdio.h>
```

```
int main(void)
```

```
{
```

```
    int x[ ]={1,3,5,7,9,11,13,15,17,19}, index;
```

```
    index=BinarySearch(x,11,10);
```

```
    if(index!=-1)
```

```
        printf("find %d!\n",x[index]);
```

```
    else
```

```
        printf("not find!\n");
```

```
    return 0;
```

```
}
```

运行结果是：
find 11!



7.3 字符数组

- 类型为char的一维数组
- 存放字符数据
- 字符串 - - - 用字符数组存放
- 在末尾加一个空字符 ‘\0’ 来构造字符串
- 字符串常量 “……” 隐含有空字符 ‘\0’
- %s 也会自动加 ‘\0’
- 字符串的长度 = 字符串的存储长度-1



7.3.1 字符数组的声明和使用

```
char s[81];
```

s是字符型数组，可以放入81个字符

可以表示最多由80个字符组成的字符串

字符数组的最小长度=该字符串的长度+1

例 编程产生26个大写英文字母的字符串 和26个小写英文字母的字符串

```
#include <stdio.h>
void main(void)
{
    char Capital[27], Lowercase[27];
    int i ;
    for(i=0; i<26; i++) {
        Capital[i]='A'+i;
        Lowercase[i]='a'+i;
    }
    Capital[26]= Lowercase[26] ='\0'; /*在末尾加 '\0' 来构造字符串
    */
    puts(Capital);
    puts(Lowercase);
}
```

7.3.2 字符数组的初始化

- 通过初始化列表(直接给出字符串中的各字符)

```
char s1[8]={ 'W', 'u', 'h', 'a', 'n', '\0' };
```

- `\0` 必须在初始化列表中显示给出
- 当数组长度未给出并且无 `\0` 时 ?

- 指定一个字符串常量 (常用)

```
char s2[28]= "Computer Science" ;
```

- 末尾将自动加上一个 `\0`

当数组长度未给出时?

```
char s3[] = "Computer" ;
```

- 字符数组的长度等于字符串的存储长度



7.4 字符串处理函数

- 串操作函数 (**<string.h>**)
 - 求字符串长度(**strlen**),(不计 ‘\0’)
 - 字符串的拷贝(**strcpy**)
 - 字符串的比较(**strcmp**)
 - 字符串的连接(**strcat**)
 - 求字符串的子串(**strstr**)
 - 删除字符串首尾空白字符
 - 从字符串中删除所有与给定字符相同的字符
 - 将字符串反转等函数(**strrev**)

例 求字符串长度的函数

/ gets length of s */*

```
int mystrlen(char s[ ])
```

```
{   int j=0;
```

```
    while(s[j]!= '\0' )    j++;
```

```
    return j;
```

```
}
```

```
int main(void)
```

```
{ char str[100];
```

```
    gets(str); /* scanf("%s",str); */
```

```
    printf("length of the string is %d\n",mystrlen(str));
```

```
    return 0;
```

```
}
```

将字符串反转的函数

交换t[i]与t[j]

t:

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

↑
i

↑
j

t:

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

↑
i

↑
j

t:

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

↑↑
ij

i>=j 结束

```
void mystrev(char s[])
{
    int i,j; /*前、后指示器*/
    char c;
    for(i=0,j=strlen(s)-1;i<j;i++,j--)
    {
        c=s[i];
        s[i]=s[j];
        s[j]=c;
    }
}
```

字符串拷贝的函数

/ copies string s to t */*

```
void mystrcpy(char t[ ], char s[ ])
```

```
{
```

```
    int j=0;
```

```
    while( (t[j]=s[j]) !='\0' ) /* 可为 !=0 , 或去掉 !='\0' */
```

```
        j++;
```

```
}
```

```
int main(void)
```

```
{    char str1[30], str2[]= "there is a boat on the lake.";
```

```
    mystrcpy(str1,str2);
```

```
    puts(str1);
```

```
    return 0;
```

```
}
```

运行结果:

there is a boat on the lake.

字符串拷贝的函数

/ copies string s to t */*

void mystrcpy(char t[], char s[]) 对吗?

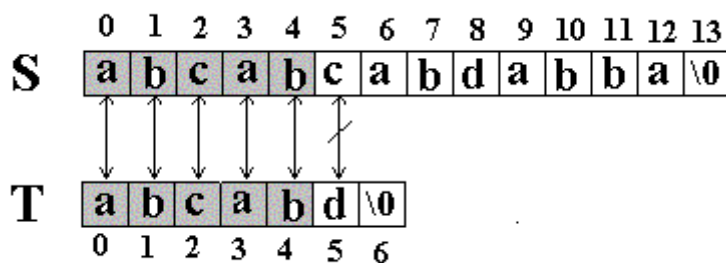
```
{
    int j=0;
    while( (s[j] !='\0' )
        t[j]=s[j++];
}

int main(void)
{
    char str1[30]="there", str2[]= "is";
    mystrcpy(str1,str2);
    puts(str1);
    return 0;
}
```

两个字符串比较函数

比较规则

从两个字符串的第一个字符起开始，按照字符**ASCII**码值的大小进行比较。



```
/* compares t to s  
return a value: =0 if s=t  
                >0 if s>t  
                <0 if s<t */
```

```
int mystrcmp(char s[],char t[])  
{  
    int j=0;  
    while(s[j]==t[j] && s[j]!='\0')  
        j++;  
    return s[j]-t[j];  
}
```


例 验证密码

```
int main(void)
{
    char pw[]="1234",s[20];
    int count=3;
    do {
        printf("Input password\n") ;
        scanf("%s",s);
        count--;
    } while(mystrcmp(pw,s)&&count));
    if(mystrcmp(pw,s)
        return 1;
        .... // 进入系统
    return 0;
}
```

字符串的连接函数

/ appends s to t */*

```
char * mystrcat(char t[],char s[])
```

```
{   int j=0,k=0;
    while(t[j++]!='\0') ;
    j--;
    while((t[j++]=s[k++])) ;
    return t;
}
```

```
void main(void)
```

```
{   char s1[80]="I like ", s2[]="the C programming. ";
    mystrcat(s1,s2);
    printf("%s\n",s1);
}
```

运行结果：

I like the C programming.



```
/* appends s to t */
```

```
char * mystrcat(char t[],char s[])
```

```
{   int j=0,k=0;
    while(t[j++]!='\0') ;
    /*j--;*/          怎样?
    while((t[j++]=s[k++]));
    return t;
}
```

运行结果:
I like

```
void main(void)
```

```
{   char s1[80]="I like ", s2[]="the C programming. ";
    mystrcat(s1,s2);
    printf("%s\n",s1);
}
```



```
/* appends s to t */
```

```
char * mystrcat(char t[],char s[])
```

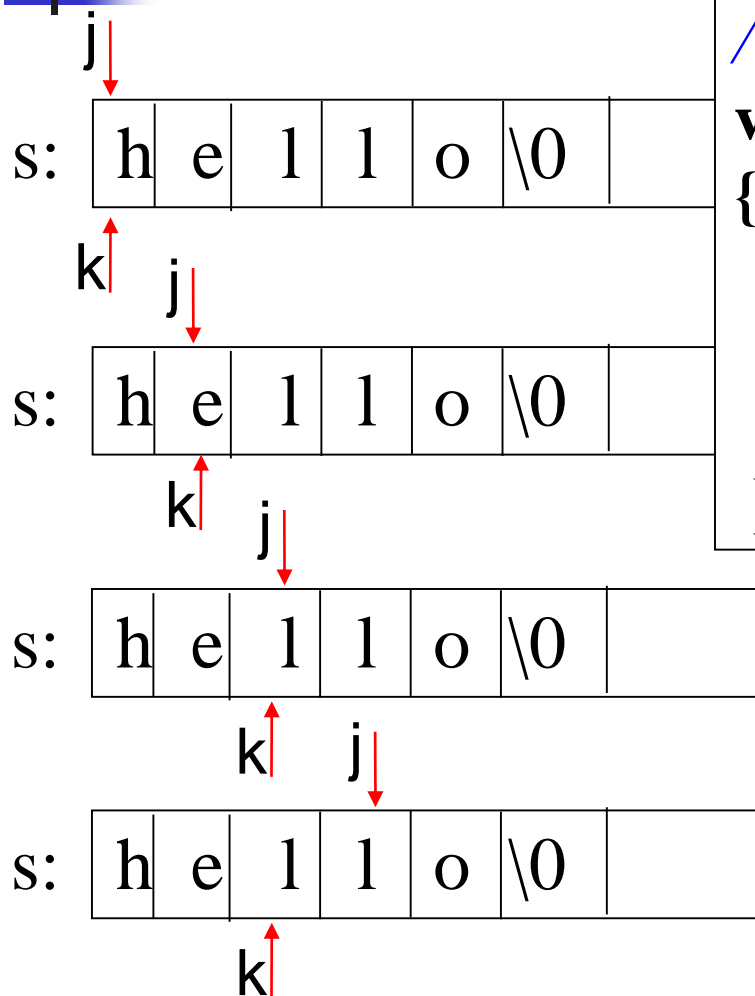
```
{   int j=0,k=0;
    while(t[j++]!='\0') ;
    j--;
    while((t[j++]=s[k++]));
    return t;
}
```

```
void main(void)
```

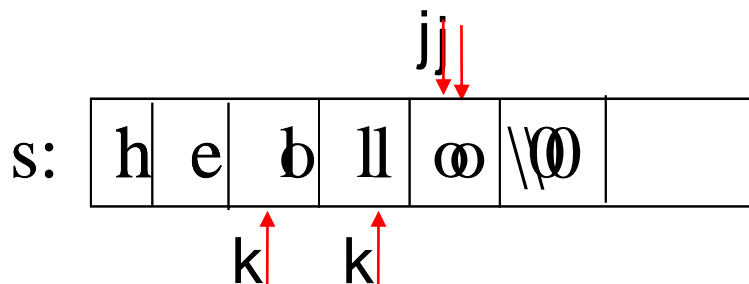
```
{   char s1[ ]="I like ", s2[]="the C programming. ";
    mystrcat(s1,s2);
    printf("%s\n",s1);
}
```

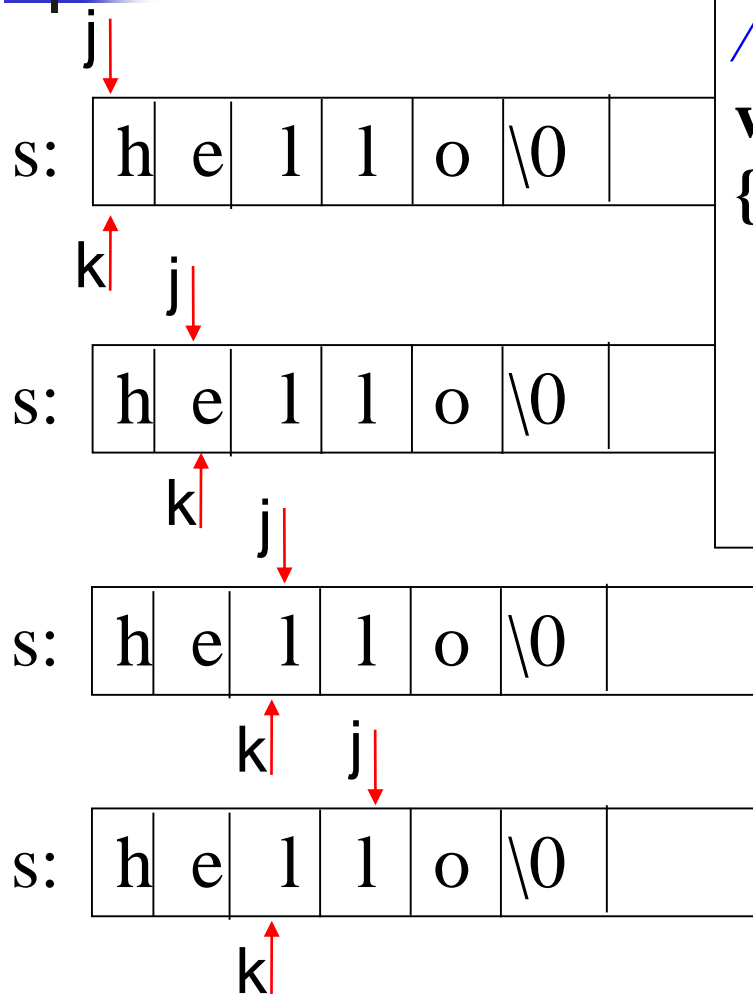
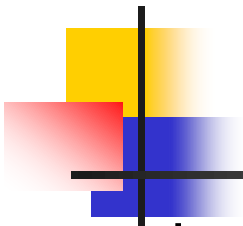
缺省长度？

从串s中删除所有与给定字符c相同的字符



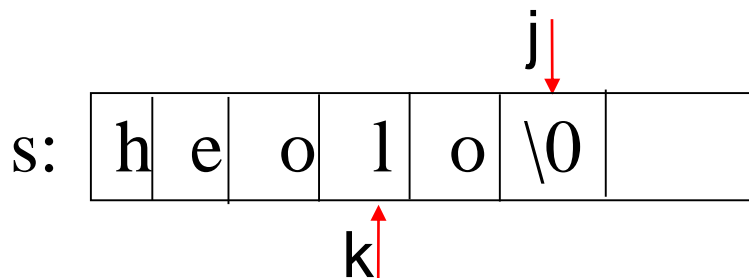
```
/* deletes all character c in s */
void strdelc(char s[ ],char c)
{ int j, k ;
  for(j=k=0;s[j]!='\0' ;j++)
    if(s[j]!=c) s[k++]=s[j];
  s[k]='\0' ;
}
```

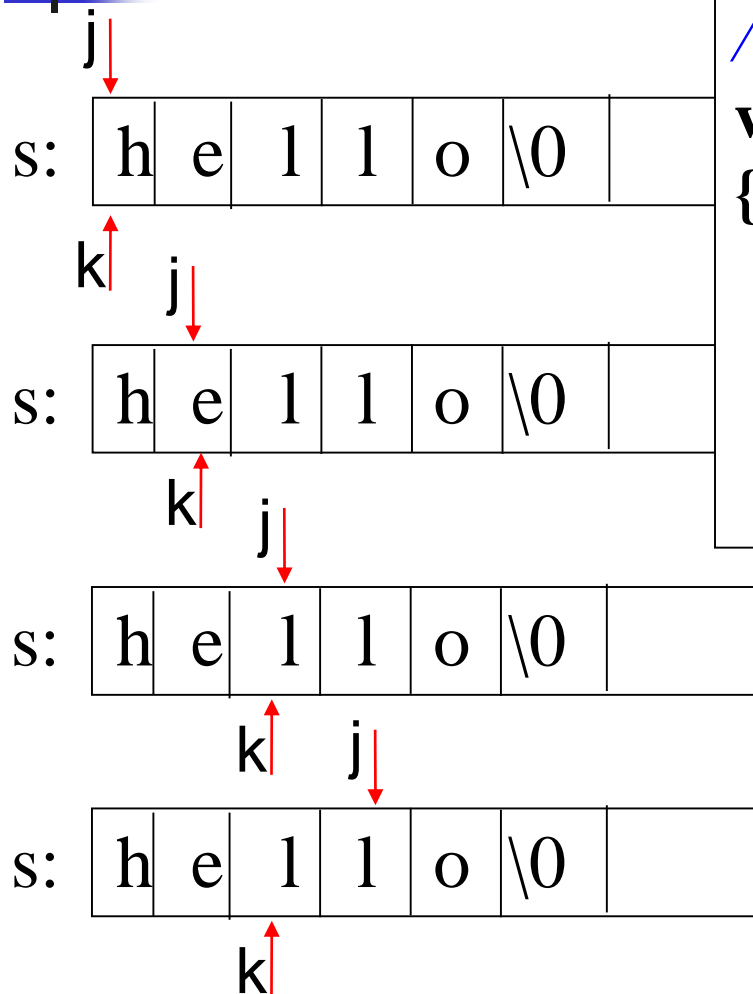
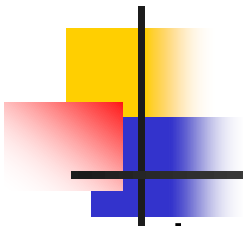




/ deletes all character c in s */*

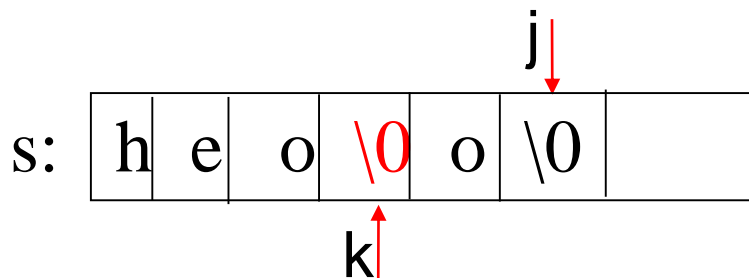
```
void strdelc(char s[],char c)
{ int j, k ;
  for(j=k=0;s[j]!='\0' ;j++)
    if(s[j]!=c) s[k++]=s[j];
  s[k]='\0' ;
}
```





/ deletes all character c in s */*

```
void strdelc(char s[],char c)
{ int j, k ;
  for(j=k=0;s[j]!='\0';j++)
    if(s[j]!=c) s[k++]=s[j];
  s[k]='\0';
}
```





7.4.2 数字串与数之间转换的函数

- 串与数值之间的转换函数（<stdlib.h>）
 - 十进制串转换为整数(atoi)
 - 十进制串转换为长整数(atol)
 - 十进制串转换为浮点型数(atof)
 - 整数转换为十进制串(itoa)
 - 十六进制串转换为整数(atoi)（无）



十进制串转换为整数 (atoi)

将一个十进制数字串转换为对应的整数的函数

- **atoi函数功能**

- 将s字符数组中存放的一个十进制数字串转换为对应的整数，并返回该整数

- **算法：**

- **ASCII码字符s[j]转换为对应数字**

- $s[j] - '0'$

- **本位乘以10加下一位的算法**

- $54321 = ((((5) * 10 + 4) * 10 + 3) * 10 + 2 * 10) + 1$



atoi函数

```
#define BASE 10
```

```
/* converts a string s to an int */
```

```
int atoi(char s[])
```

```
{
```

```
    int j=0,num=0;
```

```
    for(;s[j]!='\0';j++)
```

```
        num=num*BASE +s[j]-'0';
```

```
    return num;
```

```
}
```

将一个整数转换为十进制数字串的函数

函数名: *myitoa*

形式参数:

待转换的整数: *n*

转换之后的数字串怎么返回? 函数不能返回
(*return*) 数组类型。

转换结果串: 字符数组*s*

返回值: 无

➤ 函数原型

void myitoa(int n, char s[]);

➤ 算法:

分解出*n*的每一位数字存于数组*s*中。



```
#define BASE 10
```

```
/* converts a integer n to a string s */
```

```
void myitoa(int n, char s[])
```

```
{   int sign, j=0;
```

```
    if((sign=n)<0) n=-n;
```

```
    while(n>0){
```

```
        s[j++] = n%BASE + '0';
```

```
        n/=BASE;
```

```
    }
```

```
    if(sign<0)
```

```
        s[j++] = '-';
```

```
    s[j] = '\0';
```

```
    strrev(s);
```

```
}
```



将一个以0x（0X）为前缀的十六进制数字串转换为对应的整数的函数

■ 问题

- 当基数BASE大于10，如：16，由于十六进制数的表示形式，如'a'和'A'，'b'和'B'
- 以及0到f或F在ASCII码表的编码不连续性
- 因此需要在转换中进行一定的调整

■ htoi函数

- 将一个存放在字符数组s中的十六进制数字串转换为对应的整数
- 并且返回转换后的整数

```
int htoi(char s[])
```

```
{ int j=0,num=0;
```

```
    if(s[j]== '0'    && (s[j+1]== 'x'    || s[j+1]== 'X' )) /*跳过  
        前缀*/
```

```
        j+=2;
```

```
    else    return-1;    /* 不是十六进制数字串 */
```

```
    for(;s[j]!='\0';j++){
```

```
        if(s[j]>='0' && s[j]<='9') num=num*16+s[j]-'0';
```

```
        if(s[j]>='a' && s[j]<='f') num=num*16+s[j]-'a'+10;
```

```
        if(s[j]>='A' && s[j]<='F') num=num*16+s[j]-'A'+10;
```

```
    }
```

```
    return num;
```

```
} /* 不能处理前导空白符和符号位 ， 如何改进？
```

```
    " 0XBE" ,    " -0XBE" */
```

7.5 二维数组

- 有两个下标
- 可用于表示数据阵列(二维数据表格)

	语文	数学	英语	物理
01	85	91	80	78
02	82	95	88	92
...

如何描述上面的课程成绩表中的成绩数据？

用二维数组

```
int score[30][4]; /* score是30行4列的int型数组 */
```

二维数组还可以描述数学中的矩阵或行列式



7.5.1 二维数组的声明与使用

■ 说明

存储类 类型名 数组名[常量表达式][常量表达式]

■ 使用

数组名[行下标][列下标]

例：设全班同学修了高等数学、普通物理、c语言和英语4门课程并取得了成绩，要求计算每个同学的平均成绩，并输出成绩表。

序号	数学	物理	c	英语	平均成绩
01	91	78	91	80	78
02	95	92	95	88	92

用二维数组表示上表。

```
int s[N][M+1]; /* N-人数，M-课程数*/
```

```

#include<stdio.h>
#define N 10      /* 人数 */
#define M 4      /* 课程数 */
double CalAverage (int [ ], int ) ;    /* 该函数定义见7.1.3 */
int main(void)
{
    int x[N][M+1];
    int i,j;
    for(i=0;i<N;i++){
        for(j=0;j<M;j++)
            scanf("%d",&x[i][j]);    /*输入成绩*/
        x[i][M]=CalAverage (x[i], M) ; /*计算学生的平均成绩*/
    }
    printf("\n");
    for(i=0;i<N;i++){ /*输出成绩表*/
        for(j=0;j<=M;j++)
            printf("%d\t",x[i][j]);
        printf("\n");
    }
    return 0;
}

```

$x[i] \Leftrightarrow \&x[i][0]$

// 扩展程序功能：加一行，储存每门课的平均成绩及总平均成绩

7.5.2 二维数组的存储结构

二维数组的存放方式：
按行存放

`short x[2][3];`

二维数组x的逻辑结构

<code>x[0][0]</code> =85	<code>x[0][1]</code> =91	<code>x[0][2]</code> =88
<code>x[1][0]</code> =82	<code>x[1][1]</code> =95	<code>x[1][2]</code> =88

二维数组x的物理存储结构

	• • •	
0xffd0	85	<code>x[0][0]</code>
0xffd2	91	<code>x[0][1]</code>
0xffd4	88	<code>x[0][2]</code>
0xffd6	82	<code>x[1][0]</code>
0xffd8	95	<code>x[1][1]</code>
0xffda	88	<code>x[1][2]</code>
	• • •	

7.5.3 二维数组的初始化

- **按照物理存储结构赋初值**（当一维数组处理）

```
int a[2][2]={85,91,82,95};
```

第1维大小的说明有时可以省略

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

- **按照逻辑结构**（按行）**赋初值**

可读性好

```
int x[2][3]={ { 85,91,0}, {82,95, 0}};
```

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

7.5.4 二维字符数组

使用二维字符数组可以产生字符串数组

```
char text[25][80];
```

数组**text**可以存放**25**个字符串，每个字符串的最大长度为**79**.

■ 初始化

■ 与其它二维数组类似

```
char s[2][4]={ 'a','b','c','\0','d','e','f','\0'};
```

```
char s[2][4]={{'a','b','c','\0'},{'d','e','f','\0'}};
```

■ 用字符串对二维数组进行初始化（常用）

```
char devices[3][12]={ "hard disk" ," CRT" ," keyboard" };
```

```
<=> char devices[ ][12]={ "hard disk" ," CRT" ," keyboard" };
```

二维字符数组的使用

```
char devices[ ][12]={ "hard disk", "CRT", "keyboard"};
```

h	a	r	d		d	i	s	k	\0		
C	R	T	\0								
k	e	y	b	o	a	r	d	\0			

- 可以引用单个字符元素

`devices[2][3]` 值为 'b'

- 可以引用字符串

`devices[i]`表示`devices`数组中第*i*行字符串的首地址

```
printf( "%s" , devices[1]); /* 输出 CRT */
```

字符串数组的输入输出操作

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int i;
```

```
    char devices[ ][12]={"hard disk", "CRT", "keyboard"};
```

```
    devices[0][0]='H';    /* "hard disk"变为"Hard disk"*/
```

```
    devices[2][0]='K';    /* "keyboard "变为"Keyboard "*/
```

```
    for(i=0;i<3;i++)
```

```
        printf("%s\n", &devices[i][0]); /* 同devices[i] */
```

```
    scanf("%s",devices[1]); /* 同 &devices[i][0] */
```

```
    for(i=0;i<3;i++)
```

```
        printf("%s\n", devices[i]);
```

```
    return 0;
```

```
}
```

7.7 三维数组

■ 三维数组的用途

三维数组可以描述空间中的点集

n维数组来描述n维线性空间中的n维向量

```
int a[ ][3][3]={ { {1,2,0},{3,4,0},{5,0,0} },  
                  { {0,6,7},{8,0,0},{9,0,0} }  
                };
```

第1页

0	6	7
8	0	0
9	0	0

第0页

1	2	0
3	4	0
5	0	0