

第 6 章

利用数组处理批量数据

1. 主要内容

- (1) 引入数组的原因
- (2) 数组定义及使用（一维数组、二维数组）
- (3) 字符数组和字符串
- (4) 基于数组的简单算法（查找、排序）

2. 基本要求

- (1) 熟悉数组的含义及在内存中的表示
- (2) 掌握数组定义及使用方法
- (3) 掌握数组相关的常见算法（查找、排序等）
- (4) 掌握数组作为函数参数的本质和使用方法
- (5) 掌握字符数组与字符串的区别以及字符数组使用方法
- (6) 熟悉常用字符串处理库函数的使用方法

3. 重点、难点

重点：声明数组和引用数组的语法；数组作为函数参数的本质和使用方法；字符串的特殊性；字符串操作库函数的使用。

难点：数组的含义及在内存中的表示；数组作为函数参数的本质和使用方法；数组相关的常见算法。

4. 作业及课外学习要求

作业：编写2-4个数组相关的程序

课外学习要求：通过上机练习掌握数组定义方法和使用方法，了解一些可以用数组解决的实际问题



为什么需要数组

- 要向计算机输入全班50个学生一门课程的成绩

解决方法

用50个float型简单变量表示学生的成绩



- 烦琐，如果有1000名学生怎么办呢？
- 没有反映出这些数据间的内在联系，实际上这些数据是同一个班级、同一门课程的成绩，它们具有相同的属性。

数组

S₁₅ → s[15]

- (1) 数组是一组有序数据的集合。数组中各数据的排列是有一定规律的，下标代表数据在数组中的序号。
- (2) 用数组名和下标即可唯一地确定数组中的元素。
- (3) 数组中的每一个元素都属于同一个数据类型。

定义一维数组

类型说明符 数组名[常量表达式]

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

整型数组，即数组中的元素均为整型

数组名为a

数组包含10个整型元素

(1) **数组名**的命名规则和变量名相同，遵循标识符命名规则。

(2) 在定义数组时，需要指定数组中元素的个数，方括号中的常量表达式用来表示元素的个数，即**数组长度**。

(3) 常量表达式中可以包括常量和符号常量，不能包含变量。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
------	------	------	------	------	------	------	------	------	------

相当于定义了10个简单的整型变量

注意

- 数组元素的**下标从0开始**，用“int a[10];”定义数组（10个整型元素），则最大下标值为9，不存在数组元素a[10]



引用一维数组元素

数组名[下标]

只能引用数组元素而不能一次整体调用整个数组全部元素的值。

数组元素与一个简单变量的地位和作用相似。

“下标”可以是整型常量或整型表达式。

注意

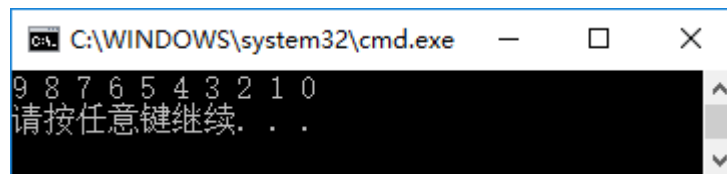
- 定义数组时用到的“数组名[常量表达式]”和引用数组元素时用的“数组名[下标]”形式相同，但含义不同。

```
int a[10];  
//前面有int, 这是定义数组, 指定数组包含10个元素  
  
t = a[6];  
//这里的a[6]表示引用a数组中序号为6的元素（第1  
个元素a[0]）
```

引用一维数组元素

【例6.1】对10个数组元素依次赋值为0,1,2,3,4,5,6,7,8,9，要求按逆序输出。

```
#include<stdio.h>
int main()
{
    int i,a[10];
    for(i=0; i<=9;i++)    //对数组元素a[0]~a[9]赋值
        a[i]=i;
    for(i=9;i>=0;i--)    //输出a[9]~a[0]共10个数组元素
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
9 8 7 6 5 4 3 2 1 0
请按任意键继续...
```



第1个for循环使a[0]~a[9]的值为0~9。

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	1	2	3	4	5	6	7	8	9

第2个for循环按a[9]~a[0]的顺序输出各元素的值。

一维数组的初始化

为了使程序简洁，常在定义数组的同时给各数组元素赋值，这称为数组的**初始化**。

(1) 在定义数组时对全部数组元素赋予初值。

```
int a[10]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

将数组中各元素的初值顺序放在一对花括号内，数据间用逗号分隔。花括号内的数据就称为“**初始化列表**”。

(2) 可以只给数组中的一部分元素赋值。

```
int a[10]={0, 1, 2, 3, 4};
```

定义a数组有10个元素，但花括号内只提供5个初值，这表示只给前面5个元素赋初值，系统自动给后5个元素赋初值为0。

(3) 给数组中全部元素赋初值为0。

```
int a[10]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

或

```
int a[10]={0}; //未赋值的部分元素自动设定为0(不推荐)
```

(4) 在对全部数组元素赋初值时，由于数据的个数已经确定，因此可以不指定数组长度。

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

或

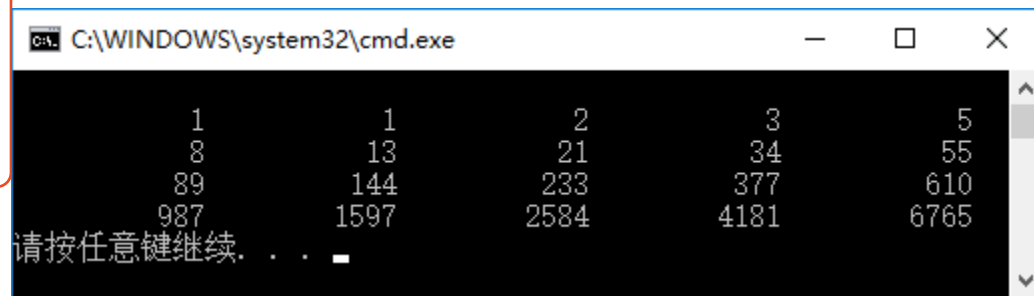
```
int a[ ]={1, 2, 3, 4, 5};
```

但是，如果数组长度与提供初值的个数不相同，则方括号中的数组长度不能省略。

一维数组程序举例

【例6.2】用数组来处理求Fibonacci数列问题。

```
#include <stdio.h>
int main()
{
    int i;
    int f[20]={1,1};           //对最前面两个元素f[0]和f[1]赋初值1
    for(i=2;i<20;i++)
        f[i]=f[i-2]+f[i-1];    //先后求出f[2]~f[19]的值
    for(i=0;i<20;i++)
    {
        if(i%5==0) printf("\n"); //控制每输出5个数后换行
        printf("%12d",f[i]);      //输出一个数
    }
    printf("\n");
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe

1      1      2      3      5
8      13     21     34     55
89     144    233    377    610
987    1597   2584   4181   6765
请按任意键继续. . .
```




一维数组程序举例

【例6.3】有10个地区的面积，要求对它们按由小到大的顺序排列。



9	9
8	8
5	5
4	4
2	2
0	0

原始数据

第一趟

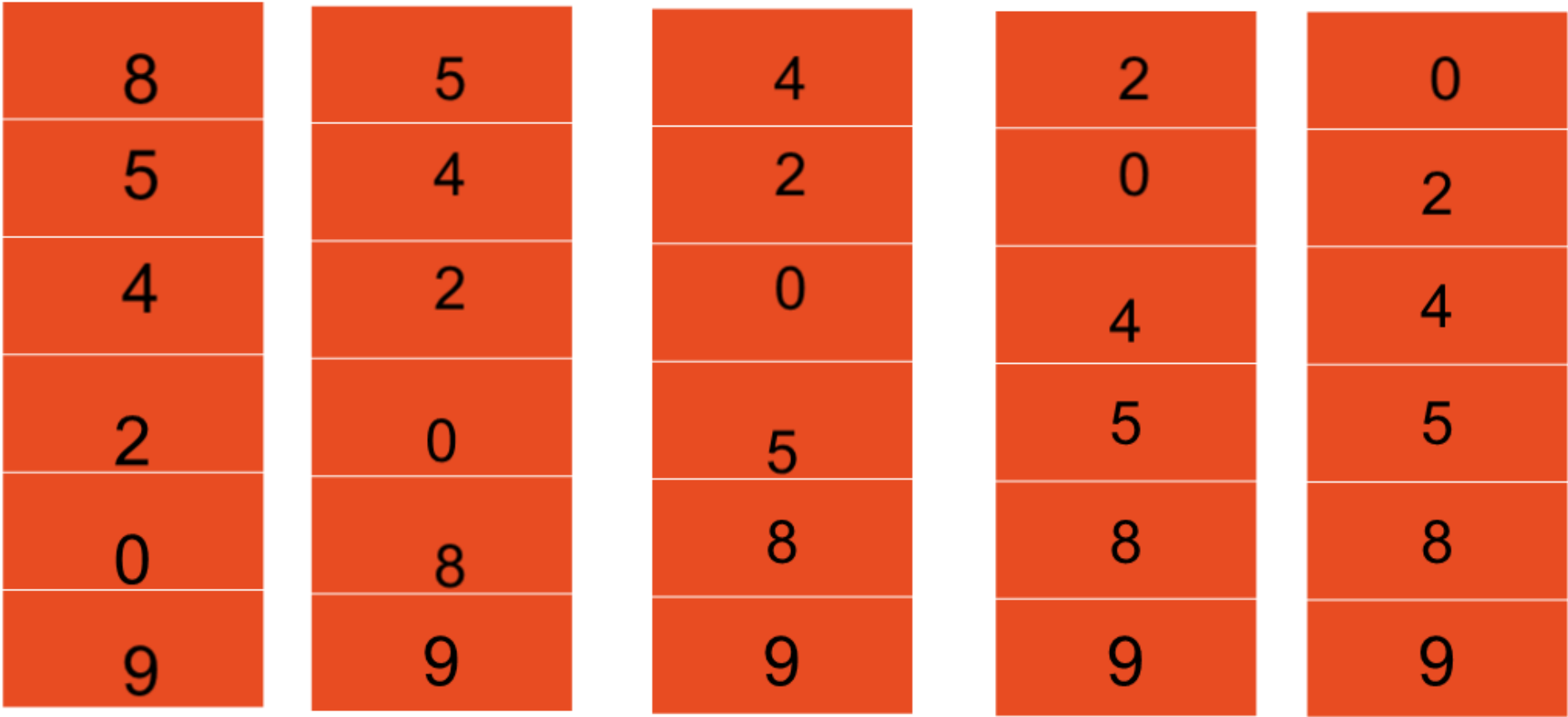


冒泡排序总结

n: 表示n个元素
 $j=1, 2, \dots, n-1$
表示第j趟排序，相邻元素两两比较，必要时交换。

第j趟排序进行n-j次相邻元素两两比较；

最多进行n-1趟排序。



第一趟

第二趟

第三趟

第四趟

第五趟

优化：第j趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有 ([n-1-j]) 趟排序。

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

```
int main()
```

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

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

```
printf("input 10 numbers :\n");
```

```
for (i=0;i<10;i++)
```

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

```
printf("\n");
```

```
for(j=0;j<9;j++)
```

//进行9次循环，实现9趟比较

```
for(i=0;i<9-j;i++)
```

//在每一趟中进行9-j次比较

```
if(a[i]>a[i+1])
```

//相邻两个数比较

```
{t=a[i];a[i]=a[i+1];a[i+1]=t;}
```

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

```
for(i=0;i<10;i++)
```

```
printf("%d ",a[i]);
```

```
printf("\n");
```

```
return 0;
```

```
}
```

【例6.3】有10个地区的面积，要求对它们按由小到大的顺序排列。

输入10个数给a[0]~a[9]

j由0变到8共执行9次循环

进行9-j次比较

a[i]>a[i+1]

真

假

a[i]与
a[i+1]交换

输出a[0]~a[9]

n: 表示n个元素

j=1, 2, .. n-1

表示第j趟排序，相邻元素两两比较，必要时交换。

第j趟排序进行n-j次相邻元素两两比较；

最多进行n-1趟排序。

```
C:\WINDOWS\system32\cmd.exe
input 10 numbers :
34 67 90 43 124 87 65 99 132 26

the sorted numbers :
26 34 43 65 67 87 90 99 124 132
请按任意键继续. . .
```

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

```
int main()
```

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

```
    int i, j, t, flag; // 标志变量flag表示第j趟排序是否进行了相邻元素  
    的交换
```

```
    printf("input 10 numbers :\n");
```

```
    for (i=0;i<10;i++)
```

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

```
    printf("\n");
```

```
    for(j=0;j<9;j++)
```

//进行9次循环，实现9趟比较

```
{ flag = 0; // 每趟排序，初始化flag，表示未进行交换
```

```
    for(i=0;i<9-j;i++) //在每一趟中进行9-j次比较
```

```
        if(a[i]>a[i+1]) //相邻两个数比较
```

```
            {t=a[i];a[i]=a[i+1];a[i+1]=t; flag = 1}
```

```
    if(!flag) break; // 表示第j趟未交换，排序好了！
```

```
}
```

```
    return 0;
```

```
}
```

【例6.3】有10个地区的面积，要求对
它们按由小到大的顺序排列。

输入10个数给a[0]~a[9]

j由0变到8共执行9次循环

进行9-j次比较

真 $a[i] > a[i+1]$ 假

a[i]与
a[i+1]交换

输出a[0]~a[9]

n: 表示n个元素

j=1, 2, .. n-1

表示第j趟排序，相
邻元素两两比较，
必要时交换。

第j趟排序进行n-j
次相邻元素两两比
较；

最多进行n-1趟排序。

优化：第j趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有
有必要进行此后的 (n-1-j) 趟排序。

```

#define MAX 100
int main()
{
    int a[MAX] = {3, 2, 1};
    int n = 3; // 实际数组大小
    int i, j, t, flag; // 标志变量flag
    scanf("%d", &n);
    for (i=0; i < n; i++) scanf("%d", &a[i]);
    for (j = 1; j <= n-1; j++) // 进行n-1次循环，实现n-1趟比较
    {
        flag = 0;
        for (i=0; i < n-j; i++) // 在每一趟中进行n-j次比较
            if (a[i] > a[i+1]) // 相邻两个数比较
                {t=a[i]; a[i]=a[i+1]; a[i+1]=t; flag = 1;}
        printf("第%d趟排序:", j);
        for (t = 0; t < n; t++) printf("%d ", a[t]); // 临时变量t的复用
        if (!flag) break;
    }
    printf("\nthe sorted numbers :\n");
    for (i = 0; i < n; i++) printf("%d ", a[i]);
    return 0;
}

```

n: 表示n个元素

j=1, 2, ... n-1

表示第j趟排序，相邻元素两两比较，必要时交换。

第j趟排序进行n-j次相邻元素两两比较；

最多进行n-1趟排序。

优化：第j趟排序中，没有进行相邻元素的交换，表示数据已经排序好，没有必要进行此后的 (n-1-j) 趟排序。



定义和引用二维数组

小例子

有3个小分队，每队有6名队员，要把这些队员的工资用数组保存起来以备查。

	队员1	队员2	队员3	队员4	队员5	队员6
第1分队	2456	1847	1243	1600	2346	2757
第2分队	3045	2018	1725	2020	2458	1436
第3分队	1427	1175	1046	1976	1477	2018

如果建立一个数组pay，它应当是二维的，第一维用来表示第几分队，第二维用来表示第几个队员。例如用 $\text{pay}_{2,3}$ 表示2分队第3名队员的工资，它的值是1725。

二维数组常称为**矩阵** (matrix)。把二维数组写成**行** (row) 和**列** (column) 的排列形式，可以有助于形象化地理解二维数组的逻辑结构。

定义二维数组

数组名为pay

float型二维数组

类型说明符 数组名[常量表达式][常量表达式]

float
pay[3][6];



```
float a[3][4], b[5][10];  
//定义a为3×4(3行4列)的数组, b为5×10(5行10列)的数组
```

数组第二维有6个元素



```
float a[3, 4], b[5, 10]; //在一对方括号内不能写两个下标
```

数组第一维有3个元素

二维数组可被看作一种特殊的一维数组：它的元素又是一个一维数组。

例如，`float a[3][4];`可以把a看作一个一维数组，它有3个元素：`a[0]`，`a[1]`，`a[2]`，每个元素又是一个包含4个元素的一维数组：

`a[0]` —— `a[0][0]` `a[0][1]` `a[0][2]` `a[0][3]`

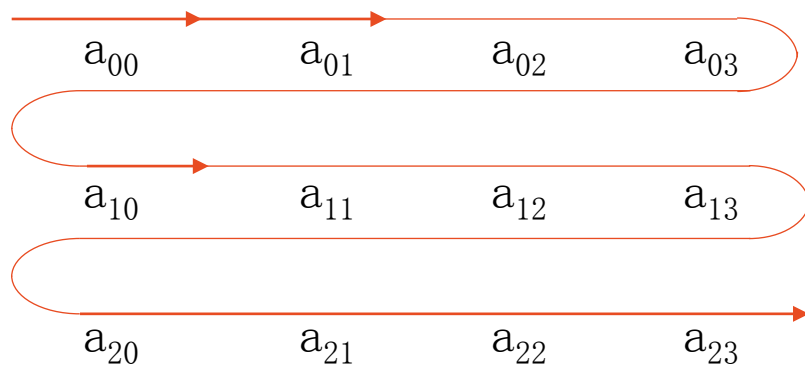
`a[1]` —— `a[1][0]` `a[1][1]` `a[1][2]` `a[1][3]`

`a[2]` —— `a[2][0]` `a[2][1]` `a[2][2]` `a[2][3]`

二维数组的存储

C语言中，二维数组中元素排列的顺序是按行存放的。

```
float a[3][4]
```



注意

- 用矩阵形式（如3行4列形式）表示二维数组，是逻辑上的概念，能形象地表示出行列关系。而在内存中，各元素是连续存放的，不是二维的，是线性的。

2000	$a[0][0]$	第0行元素
2004	$a[0][1]$	
2008	$a[0][2]$	
2012	$a[0][3]$	
2016	$a[1][0]$	第1行元素
2020	$a[1][1]$	
2024	$a[1][2]$	
2028	$a[1][3]$	
2032	$a[2][0]$	第2行元素
2036	$a[2][1]$	
2040	$a[2][2]$	
2044	$a[2][3]$	

多维数组

```
float a[2][3][4];           //定义三维数组a，它有2页，3行，4列
```

多维数组元素在内存中的排列顺序为：第1维的下标变化最慢，最右边的下标变化最快。

`float a[2][3][4];`在内存中的排列顺序为：

`a[0][0][0] → a[0][0][1] → a[0][0][2] → a[0][0][3] → a[0][1][0] → a[0][1][1]`
`→ a[0][1][2] → a[0][1][3] → a[0][2][0] → a[0][2][1] → a[0][2][2] →`
`a[0][2][3] → a[1][0][0] → a[1][0][1] → a[1][0][2] → a[1][0][3] → a[1][1][0]`
`→ a[1][1][1] → a[1][1][2] → a[1][1][3] → a[1][2][0] → a[1][2][1] →`
`a[1][2][2] → a[1][2][3]`

引用二维数组元素

数组名[下标][下标]

“下标”可以是整型常量或整型表达式。

```
int a[3][4];
```

```
int i = 0, x;
```

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

```
    printf(“%d\t”,a[i*2][0]);
```

数组元素可以出现在表达式中，也可以被赋值，如：

```
int b[1][2];
```

```
b[0,1]=a[2][3]/2;
```

注意

- 在引用数组元素时，下标值应在已定义的数组大小的范围内。

```
int a[3][4]; //定义a为3×4的二维数组
```

```
a[3][4]=3; //不存在a[3][4]元素
```

//数组a可用的“行下标”的范围为0~2，“列下标”的范围为0~3



- 严格区分在定义数组时用的a[3][4]和引用元素时的a[3][4]的区别。前者用a[3][4]来定义数组的维数和各维的大小，后者a[3][4]中的3和4是数组元素的下标值，a[3][4]代表行序号为3、列序号为4的元素（行序号和列序号均从0起算）。

二维数组的初始化

可以用“初始化列表”对二维数组初始化。

(1) 分行给二维数组赋初值。（最清楚直观）

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

(2) 可以将所有数据写在一个花括号内，按数组元素在内存中的排列顺序对各元素赋初值。

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

(3) 可以对部分元素赋初值。

```
int a[3][4]={{1}, {5}, {9}}; // ①
```

```
int a[3][4]={{1}, {0, 6}, {0, 0, 11}}; // ②
```

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

```
int a[3][4]={{1}, {}, {9}}; // ④
```

①

1 0 0 0

5 0 0 0

9 0 0 0

②

1 0 0 0

0 6 0 0

0 0 11 0

③

1 0 0 0

5 6 0 0

0 0 0 0

④

1 0 0 0

0 0 0 0

9 0 0 0

(4) 如果对全部元素都赋初值(即提供全部初始数据)，则定义数组时对第1维的长度可以不指定，但第2维的长度不能省。

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

≡

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

(5) 在定义时也可以只对部分元素赋初值而省略第1维的长度，但应分行赋初值。

```
int a[][4]={{0, 0, 3}, {}, {0, 10}};
```

【例6.4】将一个二维数组行和列的元素互换，存到另一个二维数组中。

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

```
C:\WINDOWS\system32\cmd.exe
array a:
 1  2  3
 4  5  6
array b:
 1  4
 2  5
 3  6
请按任意键继续. . .
```

```
#include <stdio.h>
int main()
{
    int a[2][3]={ {1, 2, 3}, {4, 5, 6} };
    int b[3][2], i, j;
    printf("array a:\n");
    for(i=0;i<=1;i++)    //处理a数组中的一行中各元素
    {
        for (j=0;j<=2;j++) //处理a数组中某一列中各元素
        {
            printf("%5d", a[i][j]); //输出a数组的一个元素
            b[j][i]=a[i][j]; //将a数组元素的值赋给b数组相应元素
        } // for2 end
        printf("\n");
    } // for1 end
    printf("array b:\n"); //输出b数组各元素
```

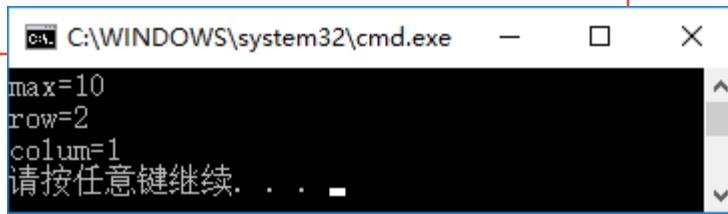
```
    for(i=0;i<=2;i++)    //处理b数组中一行中各元素
    {
        for(j=0;j<=1;j++) //处理b数组中一列中各元素
            printf("%5d", b[i][j]); //输出b数组的一个元素
        printf("\n");
    }

    return 0;
}
```

二维数组程序举例

【例6.5】有一个 3×4 的矩阵，要求编程序求出其中值最大的那个元素的值，以及其所在的行号和列号。

```
#include <stdio.h>
int main()
{
    int i, j, row=0, colum=0, max;
    int a[3][4]={1, 2, 3, 4}, {9, 8, 7, 6}, {-10, 10, -5, 2}; //定义数组并赋初值
    max=a[0][0]; //先认为a[0][0]最大
    for(i=0; i<=2; i++)
        for(j=0; j<=3; j++)
            if(a[i][j]>max) //如果某元素大于max，就取代max的原值
            {
                max=a[i][j];
                row=i; //记下此元素的行号
                colum=j; //记下此元素的列号
            }
    printf("max=%d\nrow=%d\ncolum=%d\n", max, row, colum);
    return 0;
}
```

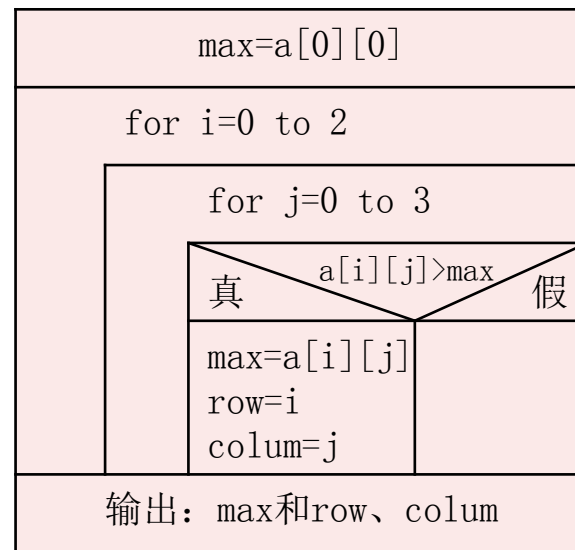


```
C:\WINDOWS\system32\cmd.exe
max=10
row=2
colum=1
请按任意键继续. . .
```

算法

找最大最小值
打擂台算法

先思考一下在打擂台时怎样确定最后的优胜者。先找出任一人站在台上，第2人上去与之比武，胜者留在台上。再上去第3人，与台上的人(即刚才的得胜者)比武，胜者留台上，败者下台。以后每一个人都是与当时留在台上的人比武。直到所有人都上台比过为止，最后留在台上的就是冠军。



字符数组



定义字符数组

用来存放字符数据的数组是**字符数组**。在字符数组中的一个元素内存放一个字符。

```
char c[10];  
c[0]=' I' ; c[1]=' ' ;c[2]=' a' ;c[3]=' m' ;c[4]=' ' ;c[5]=' h' ;c[6]=' a' ;c[7]=' p' ;c[8]=' p' ;c[9]=' y' ;
```

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
I	␣	a	m	␣	h	a	p	p	y

由于字符型数据是以整数形式(ASCII代码)存放的，因此也可以用整型数组来存放字符数据。

```
int c[10];  
c[0]=' a' ; //合法，但浪费存储空间
```



字符数组的初始化

对字符数组初始化，最容易理解的方式是用“初始化列表”，把各个字符依次赋给数组中各元素。

```
char c[10]={ ' I' , '  ' , ' a' , ' m' , '  ' , ' h' , ' a' , ' p' , ' p' , ' y' }; //把10个字符依次赋给c[0]~c[9]这10个元素
```

如果在定义字符数组时不进行初始化，则数组中各元素的值是**不可预料**的。

如果花括号中提供的初值个数（即字符个数）大于数组长度，则出现语法错误。

如果初值个数小于数组长度，则只将这些字符赋给数组中前面那些元素，其余的元素自动定为**空字符**(即 ' \0')。

```
char c[10]={ ' c' , '  ' , ' p' , ' r' , ' o' , ' g' , ' r' , ' a' , ' m' };
```

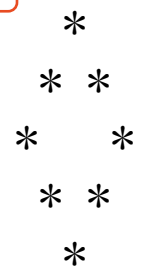
c[0]	c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]	c[8]	c[9]
c		p	r	o	g	r	a	m	\0

如果提供的初值个数与预定的数组长度相同，在定义时可以省略数组长度，系统会自动根据初值个数确定数组长度。

```
char c[]={ ' I' , '  ' , ' a' , ' m' , '  ' , ' h' , ' a' , ' p' , ' p' , ' y' };//数组c的长度自动定为10
```

也可以定义和初始化一个二维字符数组。

```
char diamond[5][5]={{'  ' , '  ' , ' *' },{'  ' , ' *' , '  ' , ' *' },{' *' , '  ' , '  ' , '  ' , '  ' , ' *' },{'  ' , ' *' , '  ' , ' *' },{'  ' , '  ' , '  ' , ' *' }};
```

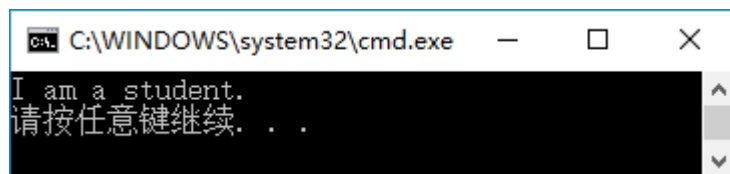


引用字符数组中的元素

空格' '

【例6.6】输出一个已知的字符串。

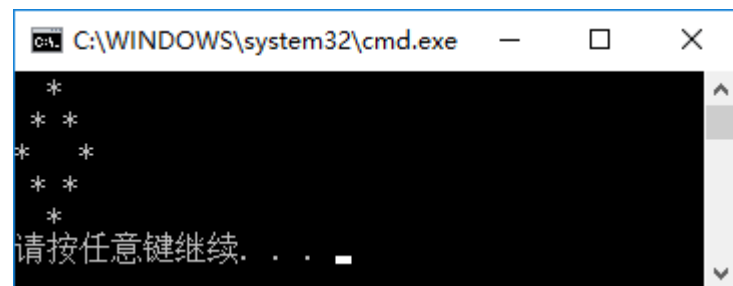
```
#include <stdio.h>
int main()
{
    char c[15]={'I', ' ', 'a', 'm', ' ', ' ', 'a', ' ', 's',
't', 'u', 'd', 'e', 'n', 't', '.'};
    int i;
    for(i=0;i<15;i++)
        printf("%c",c[i]);
    printf("\n");
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
I am a student.
请按任意键继续. . .
```

【例6.7】输出一个菱形图。

```
#include <stdio.h>
int main()
{
    char diamond[][5]={{' ', ' ', '*'}, {' ', '*', ' ',
', '*'}, {'*', ' ', ' ', ' ', '*'},
    {' ', '*', ' ', '*'}, {' ', ' ', '*'}};
    int i, j;
    for (i=0;i<5;i++)
    {
        for (j=0;j<5;j++)
            printf("%c",diamond[i][j]);
        printf("\n");
    }
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
 *
* *
*  *
* *
 *
请按任意键继续. . .
```

字符串和字符串结束标志

在C语言中，是将字符串作为字符数组来处理的。

在实际工作中，人们关心的往往是字符串的有效长度而不是字符数组的长度。

为了测定字符串的实际长度，C语言规定了一个“字符串结束标志”，以字符‘\0’作为结束标志。

```
char s[] = "C program"; // 字符串是存放在一维数组中的，占10个字节，字符占9个字节，  
                        // 最后一个字节'\0' 是由系统自动加上的
```

注意

- C系统在用字符数组存储字符串常量时会自动加一个‘\0’作为结束符。
- 在定义字符数组时应估计实际字符串长度，保证数组长度始终大于字符串实际长度。
- 如果在一个字符数组中先后存放多个不同长度的字符串，则应使数组长度大于最长的字符串的长度。

字符串和字符串结束标志

```
printf("How do you do?\n");
```

在向内存中存储时，系统自动在最后一个字符' \n' 的后面加了一个' \0' ，作为字符串结束标志。在执行printf函数时，每输出一个字符检查一次，看下一个字符是否为' \0' ，遇' \0' 就停止输出。

```
char c[]={"I am happy"};
```

或

```
char c[]="I am happy";
```

用一个字符串(注意字符串的两端是用双引号而不是单引号括起来的)作为字符数组的初值。

注意

- 数组c的长度不是10，而是11。因为字符串常量的最后由系统加上一个' \0' 。

```
char c[]={' I' , ' ' , ' a' , ' m' , ' ' , ' h' , ' a' , ' p' , ' p' , ' y' , ' \0' };
```

≠

```
char c[]={' I' , ' ' , ' a' , ' m' , ' ' , ' h' , ' a' , ' p' , ' p' , ' y' };
```

```
char c[10]={"China"};
```

数组c的前5个元素为: ' C' , ' h' , ' i' , ' n' , ' a' , 第6个元素为' \0' ，后4个元素也自动设定为空字符。

C	h	i	n	a	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----



字符数组的输入输出

- (1) 逐个字符输入输出。用格式符 “%c” 输入或输出一个字符。
- (2) 将整个字符串一次输入或输出。用 “%s” 格式符，意思是对字符串(string)的输入输出。

```
#include <stdio.h>
int main()
{
    char c[15]={' I',' ','a','m',' ','a',' ',
               's','t','u','d','e','n','t','.'};
    int i;
    for(i=0;i<15;i++)
        printf("%c",c[i]);
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char c[]="China";
    printf("%s\n",c);
    return 0;
}
```

- (1) 输出的字符中不包括结束符 ' \0' 。
- (2) 用 “%s” 格式符输出字符串时，printf函数中的输出项是字符数组名，而不是数组元素名。
- (3) 如果数组长度大于字符串的实际长度，也只输出到遇 ' \0' 结束。
- (4) 如果一个字符数组中包含一个以上 ' \0' ，则遇第一个 ' \0' 时输出就结束。



字符数组的输入输出

```
char c[6];
scanf("%s", c);
```

从键盘输入：
China✓
系统会自动在China后面加一个' \0' 结束符。

```
char str1[5], str2[5], str3[5];
scanf("%s%s%s", str1, str2, str3);
```

如果利用一个scanf函数输入多个字符串，则应在输入时以**空格**分隔。
从键盘输入：
How are you? ✓
由于有空格字符分隔，作为3个字符串输入。

str1:	H	o	w	\0	\0
str2:	a	r	e	\0	\0
str3:	y	o	u	?	\0

```
char str[13];
scanf("%s", str);
```

从键盘输入：
How are you? ✓
由于系统把空格字符作为输入的字符串之间的分隔符，因此只将空格前的字符" How" 送到str中。

H	o	w	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
---	---	---	----	----	----	----	----	----	----	----	----	----	----	----



字符数组的输入输出

注意

- scanf函数中的输入项如果是字符数组名，不要再加地址符&，因为在C语言中数组名代表该数组第一个元素的地址(或者说数组的起始地址)。



scanf("%s", &str); //str前面不应加&

- 若数组占6个字节。数组名c代表地址2000。可以用下面的输出语句得到数组第一个元素的地址。

```
char c[] = "china";  
printf("%o", c); //用八进制形式输出数组c的起始地址, 如, 22fe30  
printf("%x", c); //用十六进制形式输出数组c的起始地址, 如, 10577060
```

```
printf("%s", c); // china
```

c数组

2000	C
2001	h
2002	i
2003	n
2004	a
2005	\0

- 实际上是这样执行的：按字符数组名c找到其数组第一个元素的地址，然后逐个输出其中的字符，直到遇'\0'为止。

使用字符串处理函数

输出字符串的函数

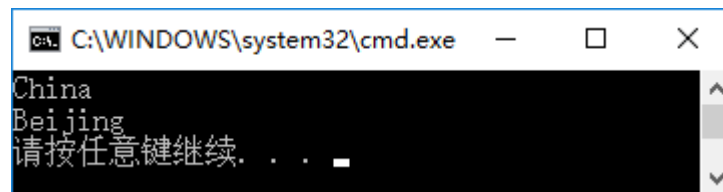
puts(字符数组)

作用：将一个字符串(以' \0' 结束的字符序列)输出到终端。

用puts函数输出的字符串中可以包含转义字符。

在用puts输出时将字符串结束标志' \0' 转换成' \n' ，即输出完字符串后换行。

```
#include <stdio.h>
int main()
{
    char str[]={"China\nBeijing"};
    puts(str);
    return 0;
}
```



输入字符串的函数

gets(字符数组)

作用：从终端输入一个字符串到字符数组，并且得到一个函数值。该函数值是字符数组的起始地址。

```
gets(str); // 可以接收带空格的字符串输入
scanf("%s", str) // 遇空格结束输入
```

```
char str[80];
```

```
gets(str); //str是已定义的字符数组
```

如果从键盘输入：

Computer✓

将输入的字符串 "Computer" 送给字符数组str（请注意，送给数组的共有9个字符，而不是8个字符），返回的函数值是字符数组str的第一个元素的地址。

注意

- 用puts和gets函数只能输出或输入一个字符串。



```
puts(str1, str2); 或 gets(str1, str2);
```

字符串连接函数

`strcat` (字符串1, 字符串2)

作用：把两个字符数组中的字符串连接起来，把字符串2接到字符串1的后面，结果放在字符数组1中，函数调用后得到一个函数值——字符数组1的地址。

字符数组1必须**足够大**，以便容纳连接后的新字符串。

连接前两个字符串的后面都有 `'\0'`，连接时将字符串1后面的 `'\0'` 取消，只在新串最后保留 `'\0'`。

```
char str1[30]={"People' s Republic of "};  
char str2[]={"China"};  
printf("%s", strcat(str1, str2));
```

输出: People's Republic of China

连接前 str1:	P	e	o	p	l	e	'	s		R	e	p	u	b	l	i	c		o	f		\0	\0	\0	\0	\0	\0	\0	\0	\0
str2:	C	h	i	n	a																									
连接后 str1:	P	e	o	p	l	e	'	s		R	e	p	u	b	l	i	c		o	f		C	h	i	n	a	\0	\0	\0	\0



字符串复制函数

```
char str1[10], str2[]="China";  
strcpy(str1, str2); 或 strcpy(str1, "China");
```

strcpy(字符数组1, 字符串2)

执行后, str1:

C	h	i	n	a	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----

作用：将字符串2复制到字符数组1中去。

字符数组1必须定义得足够大，以便容纳被复制的字符串2。字符数组1的长度不应小于字符串2的长度。

“字符数组1”必须写成数组名形式，“字符串2”可以是字符数组名，也可以是一个字符串常量。

若在复制前未对字符数组1初始化或赋值，则其各字节中的内容无法预知，复制时将字符串2和其后的‘\0’一起复制到字符数组1中，取代字符数组1中前面的字符，未被取代的字符保持原有内容。

不能用赋值语句将一个字符串常量或字符数组直接给一个字符数组。字符数组名是一个地址常量，它不能改变值，正如数值型数组名不能被赋值一样。



```
str1="China"; str1=str2;
```

可以用strncpy函数将字符串2中前面n个字符复制到字符数组1中去。

```
strncpy(str1, str2, 2);
```

将str2中最前面2个字符复制到str1中，取代str1中原有的最前面2个字符。但复制的字符个数n不应多于str1中原有的字符（不包括‘\0’）。

字符串比较函数

`strcmp(字符串1, 字符串2)`

```
strcmp(str1, str2);  
strcmp("China", "Korea");  
strcmp(str1, "Beijing");
```

作用：比较字符串1和字符串2。

字符串比较的**规则**是：将两个字符串自左至右逐个字符相比(按ASCII码值大小比较)，直到出现不同的字符或遇到' \0' 为止。

- (1) 如全部字符相同，则认为两个字符串相等；
- (2) 若出现不相同的字符，则以第1对不相同的字符的比较结果为准。

比较的**结果**由函数值带回。

- (1) 如果字符串1与字符串2相同，则函数值为0。
- (2) 如果字符串1>字符串2，则函数值为一个**正整数**。
- (3) 如果字符串1<字符串2，则函数值为一个**负整数**。

注意

- 对两个字符串比较不能直接用 `str1>str2` 进行比较，因为 `str1` 和 `str2` 代表地址而不代表数组中全部元素，而只能用 `(strcmp(str1, str2)>0)` 实现，系统分别找到两个字符数组的第一个元素，然后顺序比较数组中各个元素的值

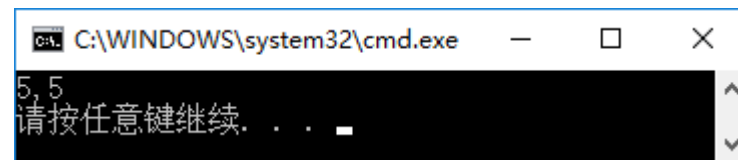


测字符串长度的函数

strlen(字符数组)

作用：测试字符串长度的函数。函数的值为字符串中的实际长度(不包括 ' \0' 在内)。

```
#include <stdio.h>
#include <string.h> // 字符串处理函数头文件
int main()
{
    char str[10]="China";
    printf("%d,%d\n", strlen(str), strlen("China"));
}
```



转换为大小写的函数

strlwr(字符串)

作用：将字符串中大写字母换成小写字母。

strupr(字符串)

作用：将字符串中小写字母换成大写字母。

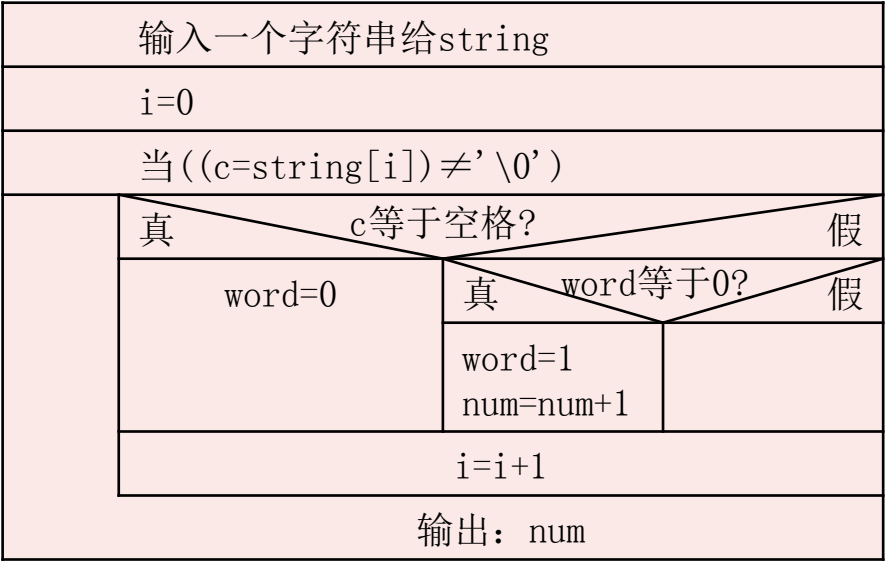
注意

- 以上介绍了常用的8种字符串处理函数，它们属于**库函数**。库函数并非C语言本身的组成部分，而是C语言编译系统为方便用户使用而提供的公共函数。不同的编译系统提供的函数数量和函数名、函数功能都不尽相同，使用时要小心，必要时查一下库函数手册。
- 在使用字符串处理函数时，应当在程序文件的开头用**#include <string.h>**把string.h文件包含到本文件中。

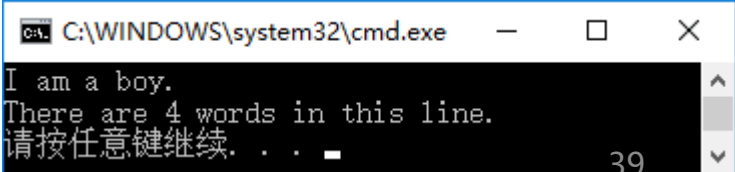
字符数组应用举例

【例6.8】输入一行字符，统计其中有多少个单词，单词之间用空格分隔开。

- string: 用于存放字符串。
- i: 计数器，用于遍历字符串中的每个字符。
- word: 用于判断是否开始了一个新单词的标志。若word=0表示未出现新单词，如出现了新单词，就把word置成1。
- num: 用于统计单词数。



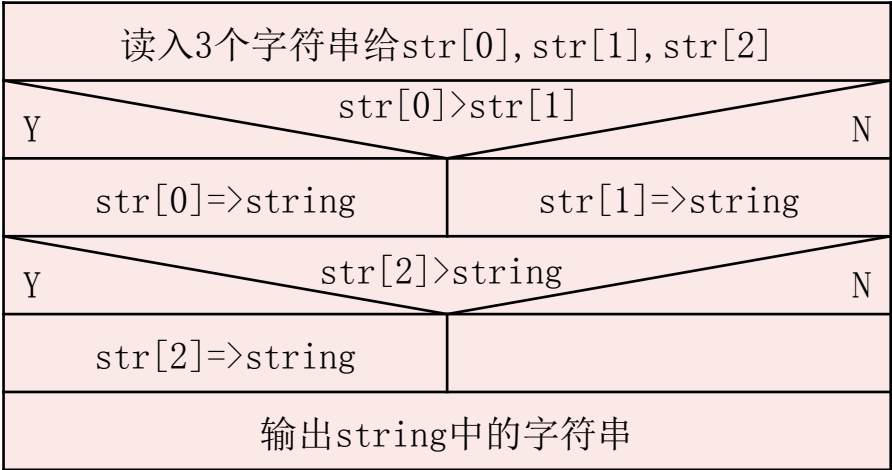
```
#include <stdio.h>
int main()
{
    char string[81];
    int i,num=0,word=0;
    char c;
    gets(string);    //输入一个字符串给字符数组string
    for(i=0;(c=string[i])!='\0';i++) //只要字符不是'\0'就循环
        if(c==' ') word=0;    //若是空格字符，使word置0
    else if(word==0) //如果不是空格字符且word原值为0
    {
        word=1;    //使word置1
        num++;    //num累加1，表示增加一个单词
    }
    printf("There are %d words in this line.\n",num); //输出单词数
    return 0;
}
```



字符数组应用举例

【例6.9】有3个字符串,要求找出其中“最大”者。

str[0]:	H	o	l	l	a	n	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[1]:	C	h	i	n	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[2]:	A	m	e	r	i	c	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0



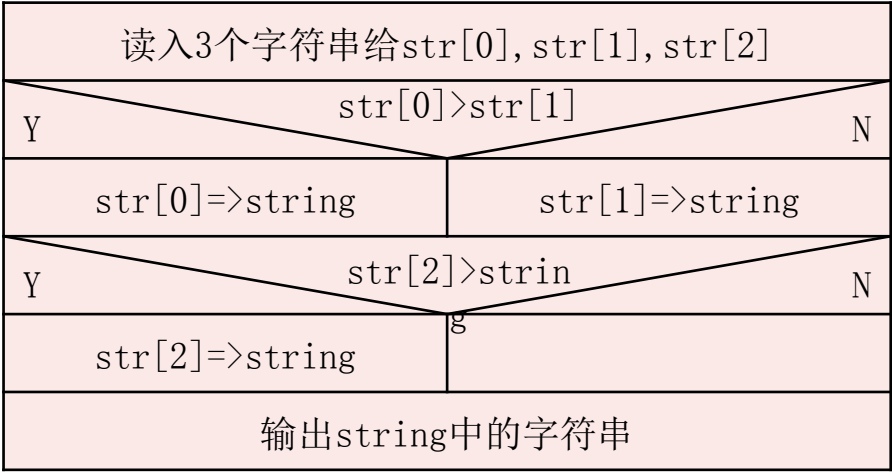
```
#include<stdio.h>
#include<string.h>
int main()
{
    char str[3][20];    //定义二维字符数组
    char string[20];
    //定义一维字符数组，作为交换字符串时的临时字符数组
    int i;
    for(i=0;i<3;i++)
        gets(str[i]); //读入3个字符串，分别给str[0], str[1], str[2]
    if(strcmp(str[0], str[1])>0) //若str[0]大于str[1]
        strcpy(string, str[0]); //把str[0]的字符串赋给字符数组string
    else
        //若str[0]小于等于str[1]
        strcpy(string, str[1]); //把str[1]的字符串赋给字符数组string
```

```
    if(strcmp(str[2], string)>0) //若str[2]大于string
        strcpy(string, str[2]); //把str[2]的字符串赋给字符数组string
    printf("\nthe largest string is:\n%s\n", string);    //输出string
    return 0;
}
```


字符数组应用举例

【例6.9】有3个字符串,要求找出其中“最大”者。

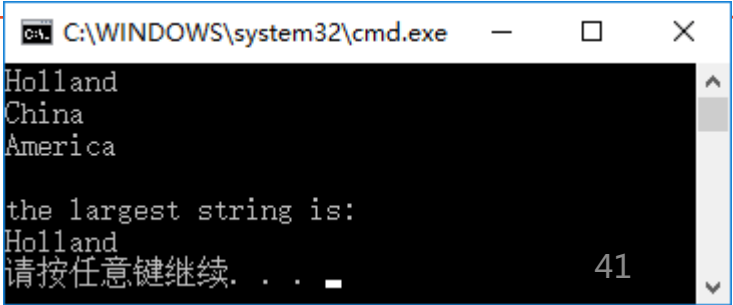
str[0]:	H	o	l	l	a	n	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[1]:	C	h	i	n	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[2]:	A	m	e	r	i	c	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0



```
#include<stdio.h>
#include<string.h>
int main()
{
    char str[3][20]; //定义二维字符数组
    char string[20];
    //定义一维字符数组, 作为交换字符串时的临时字符数组
    int i;
    for(i=0;i<3;i++)
        gets(str[i]); //读入3个字符串, 分别给str[0], str[1], str[2]
    if(strcmp(str[0], str[1])>0) //若str[0]大于str[1]
```

```
        strcpy(string, str[0]); //把str[0]的字符串赋给字符数组string
    else //若str[0]小于等于str[1]
        strcpy(string, str[1]); //把str[1]的字符串赋给字符数组string
    if(strcmp(str[2], string)>0) //若str[2]大于string
        strcpy(string, str[2]); //把str[2]的字符串赋给字符数组string
    printf("\nthe largest string is:\n%s\n", string); //输出
    string
    return 0;
}
```

- (1) 流程图和程序注释中的“大于”是指两个字符串的比较中的“大于”。
- (2) str[0], str[1], str[2]和string是一维字符数组, 其中可以存放一个字符串。
- (3) strcpy函数在将str[0], str[1]或str[2]复制到string时, 最后都有一个' \0'。因此, 最后用%s格式输出string时, 遇到string中第一个' \0' 即结束输出, 并不是把string中的全部字符输出。



注意事项小结

➤ 数组定义与引用

```
int a[10]; float b[2][3]; char string[] = "I am student!"  
a[0] = 1; a[9] = 10; b[0][0] = 1; b[1][2] = 6;  
printf( "%d,%d,%f,%f,%s\n" , a[0], a[9], b[0][0], b[1][2], string );
```

➤ 冒泡排序:

n: 表示n个元素, $j=1, 2, \dots, n$

表示第j趟排序, 相邻元素两两比较, 必要时交换;

第j趟排序进行n-j次相邻元素两两比较;

最多进行n-1趟排序。

优化: 第j趟排序中, 没有进行相邻元素的交换, 表示数据已经排序好, 没有必要进行此后的 $(n-1-j)$ 趟排序。

➤ 常用一重循环处理一维数组, 嵌套的两重循环处理二维数组。

➤ 字符串的结束标志' \0' :

```
char s1[] = "123"; char s2 = { '1' , '2' , '3' }; // s1[3] = s2[3] = '\0' ;
```

➤ 字符串的输入: include <stdio.h> scanf(); include <string.h> gets()

```
char str[80];
```

```
scanf( "%s" , str); // 遇空格或回车结束
```

```
gets(str); // 遇回车结束
```

注意事项小结

- 字符串的输出: `include <stdio.h> printf(); include <string.h> puts()`
`char str[80];`
`printf(“%s”, str); // 遇' \0' 结束`
`puts(str); // 遇' \0' 结束`
- 字符串处理函数 (所有字符串以' \0' 结束)
`include <string.h>`
`char str1[80], str2[80];`
`strlen(str1); // str1的长度(字符个数, 不包含' \0')`
`strcat(str1, str2); // str1 + str2`
`strcpy(str1, str2); // str2 copy to str1`
`strcpy(str1, str2, n) // str2的前n个字符copy to str1`
`strcmp(str1, str2); // 相同返回0; str1大于str2, 返回>0的int; str1小于str2, 返回<0的int`
`// 切记不能使用str1==str2, str1>str2, str1<str2`
`strlwr(str1); // 将str1中的大写字母转换为小写字母`
`strupr(str2); // 将str1中的小写字母转换为大写字母`