



## 4. 数据结构1——数组与自定义数据类型

授课教师：游伟 副教授

授课时间：周一08:00 – 09:30, 周四16:00 – 17:30 (明德新闻楼0201)

上机时间：周四18:00 – 21:00 (理工配楼二层5号机房)

课程主页：<https://rucsesec.github.io/programming>

# C语言运算符概览

算术运算符:	+ - * / % ++ --
赋值运算符:	= 及其扩展
求字节数 :	sizeof
强制类型转换:	(类型)
函数调用符运算符:	()
关系运算符:	< <= == > >= !=
逻辑运算符:	! &&
条件运算符:	?:
下标运算符:	[]
分量运算符:	. ->
位运算符 :	<< >> ~ &   ^
指针运算符:	* &
逗号运算符:	,

# 引子：生日祝福

【背景】为了在每位同学生生日时给同学献上祝福并且撰写人物传记，班委们不得不每天翻看通讯录寻找当日寿星。请编写一个小程序，帮助班委们完成寻找当日寿星的操作。具体而言，将每个同学在通讯录上的信息以一定组织形式存储，并在程序运行时遍历每个同学的生日信息，查看是否有当天生日的同学。

## 【思考】

- 每位同学的不同类型信息（学号、姓名、性别、生日等）如何组织在一起
- 所有同学的信息如何组织在一起
- 生日怎么存储才易于判断

# 目录

1. 一维数组
2. 二维数组
3. 多维数组
4. 字符数组
5. 枚举类型
6. 结构体
7. 共同体
8. 位域

## 4.1 一维数组

- 要向计算机输入全班50个学生一门课程的成绩

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。

解决方法



数组

为什么需要数组？

- (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。

## 4.1.1 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。

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

(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。

**注意**

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

## 4.1.2 引用一维数组

### 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]”在形式上相同，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
```

## 4.1.3 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
```

(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。



## 4.1.4 一维数组的存储

C语言中，一维数组在内存中是一块连续的区域，按下标大小依次存储。

float a[4]



2000	
2004	a[0]
2008	a[1]
2012	a[2]
	a[3]

```
1. #include<stdio.h>
2. int main()
3. {
4.     int n = 5, m = 0, l = 0;
5.     int arr1[5];
6.     int arr2[n];
7.     scanf("%d", &m);
8.     int arr3[m];
9.     int arr4[l];
10.    scanf("%d", &l);
11.    printf("%d %d %d %d\n",
12.           sizeof(arr1), sizeof(arr2),
13.           sizeof(arr3), sizeof(arr4));
14.}
```

输入:

5 5

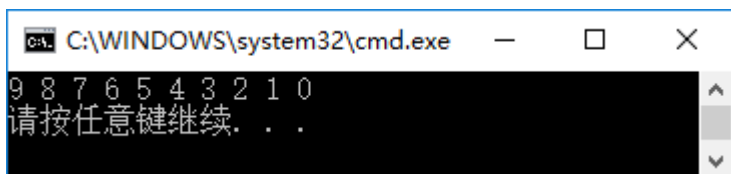
输出:

20 20 20 0

## 4.1.5 一维数组应用举例

- 【例4-1】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

```
1. #include<stdio.h>
2. int main()
3. {
4.     int i,a[10];
5.     for(i=0; i<=9;i++)           //对数组元素a[0]~a[9]赋值
6.         a[i]=i;
7.     for(i=9;i>=0;i--)           //输出a[9]~a[0]共10个数组元素
8.         printf("%d ",a[i]);
9.     printf("\n");
10.    return 0;
11.}
```



```
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。

回顾：21级图灵班选拔卷第四题【问题1】当输入为5时，下列代码的输出结果是多少？

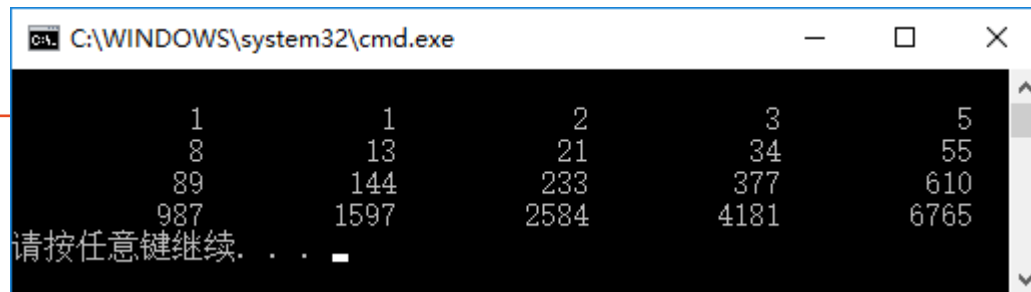
答案：8（斐波拉契数列）

```
1. INPUT n
2. x←0
3. y←1
4. FOR i : 1 to n {
5.     z←ADD(x, y)
6.     x←y
7.     y←z
8. }
9. OUTPUT z
```

## 4.1.5 一维数组应用举例

### ■ 【例4-2】用数组来处理求Fibonacci数列问题

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i;
5.     int f[20]={0,1};           //对最前面两个元素f[0]和f[1]赋初值1
6.     for(i=2;i<20;i++)
7.         f[i]=f[i-2]+f[i-1];    //先后求出f[2]~f[19]的值
8.     for(i=0;i<20;i++)
9.     {
10.        if(i%5==0) printf("\n"); //控制每输出5个数后换行
11.        printf("%12d",f[i]);     //输出一个数
12.    }
13.    printf("\n");
14.    return 0;
15.}
```



```
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. ...
```

## 4.2 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。

## 4.2.1 定义二维数组

类型说明符 数组名[常量表达式][常量表达式]

`float pay[3][6];`

float型二维数组

数组名为pay



`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



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

数组第二维有6个元素

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]`

## 4.2.2 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。

数组元素可以出现在表达式中，也可以被赋值，如：  $b[1][2] = a[2][3] / 2;$

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 $a[3][4]$ 来定义数组维数和各维大小
- 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）。

## 4.2.3 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	②	③	④
1 0 0 0	1 0 0 0	1 0 0 0	1 0 0 0
5 0 0 0	0 6 0 0	5 6 0 0	0 0 0 0
9 0 0 0	0 0 11 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};
```

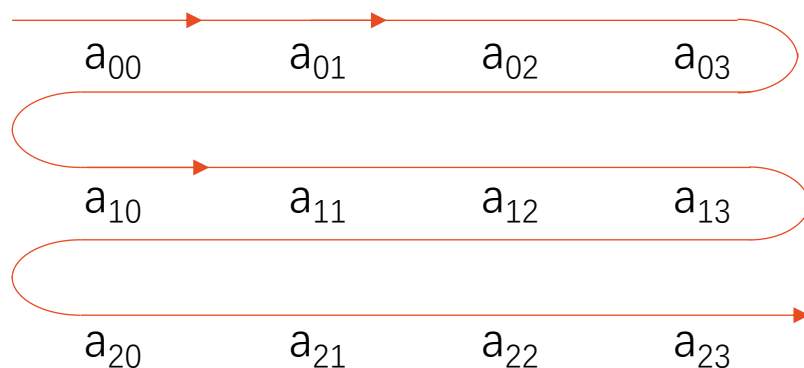
在定义时也可以只对部分元素赋初值而省略第1维的长度，但应分行赋初值。

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

## 4.2.4 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	$a[0][0]$	}
2004	$a[0][1]$	
2008	$a[0][2]$	
2012	$a[0][3]$	
2016	$a[1][0]$	}
2020	$a[1][1]$	
2024	$a[1][2]$	
2028	$a[1][3]$	
2032	$a[2][0]$	}
2036	$a[2][1]$	
2040	$a[2][2]$	
2044	$a[2][3]$	



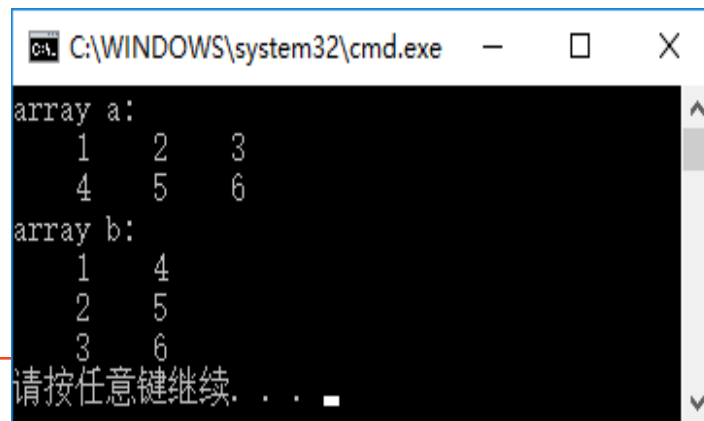
## 4.2.5 二维数组应用举例

■ 【例4-3】 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（矩阵转置）

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

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a[2][3]={{1,2,3},{4,5,6}};
5.     int b[3][2],i,j;
6.     printf("array a:\n");
7.     for(i=0;i<=1;i++) //处理a数组一行中各元素
8.     {
9.         for (j=0;j<=2;j++) //处理a数组某一列中各元素
10.        {
11.            printf("%5d",a[i][j]); //输出一个元素
12.            //将a数组元素的值赋给b数组相应元素
13.            b[j][i]=a[i][j];
14.        }
15.        printf("\n");
16.    }
```

```
17.     printf("array b:\n");
18.     for(i=0;i<=2;i++) //处理b数组一行中各元素
19.     {
20.         for(j=0;j<=1;j++) //处理b数组某一列中各元素
21.             printf("%5d",b[i][j]); //输出b数组一个元素
22.         printf("\n");
23.     }
24.     return 0;
25. }
```



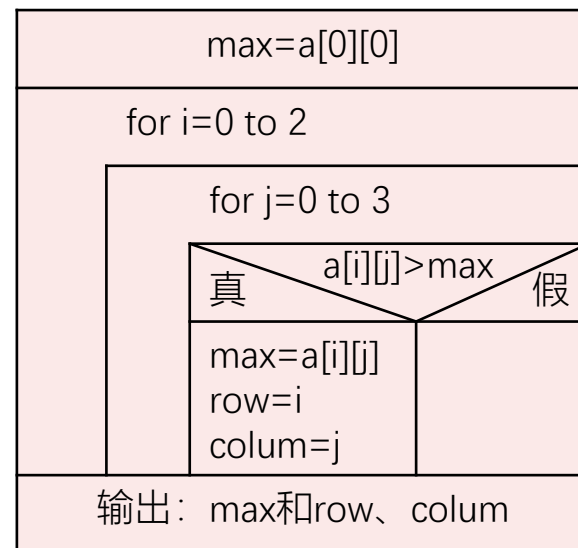
```
C:\WINDOWS\system32\cmd.exe
array a:
    1    2    3
    4    5    6
array b:
    1    4
    2    5
    3    6
请按任意键继续. . .
```

## 4.2.5 二维数组应用举例

■ 【例4-4】有一个 $3 \times 4$ 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。

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i,j,row=0,column=0,max;
5.     int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}}; //定义数组并赋初值
6.     max=a[0][0]; //先认为a[0][0]最大
7.     for(i=0;i<=2;i++)
8.         for(j=0;j<=3;j++)
9.             if(a[i][j]>max) //如果某元素大于max, 就取代max的原值
10.            {
11.                max=a[i][j];
12.                row=i; //记下此元素的行号
13.                column=j; //记下此元素的列号
14.            }
15.     printf("max=%d\nrow=%d\ncolumn=%d\n",max,row,column);
16.     return 0;
17. }
```

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。



## 4.3 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，第n维的下标变化最快。

---

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]

---

## 4.4 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]='l'; 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]
l		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，故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
```

思考：如何验证下列代码会不会在  
字符数组结尾加一个'\0'?

## 4.4.1 字符数组的初始化

```
char c[10]={ 'l', ' ', 'a', 'm', ' ', 'h', 'a', 'p', 'p', 'y'};
```

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]={ 'l', ' ', '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，定义时可以省略数组长度，系统会自动根据初值个数确定长度。

```
char c[]={ 'l', ' ', '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]={{ ' ', ' ', '*', ' ', ' '},{ ' ', '*', ' ', ' ', ' '},{ '*', ' ', ' ', ' ', '*'},{' ', '*', ' ', ' ', '*'},{' ', ' ', ' ', ' ', '*'}};
```

```
      *
    * *
 *   *
 * *
 *
```

## 4.4.2 引用字符数组中的元素

### 【例4-5】输出一个已知的字符串

```
1. #include <stdio.h>
2. int main()
3. {
4.     char c[15]={'I',' ','a','m',' ','a',' ','
5.                 's','t','u','d','e','n','t','.'};
6.     int i;
7.     for(i=0;i<15;i++)
8.         printf("%c",c[i]);
9.     printf("\n");
10.    return 0;
11.}
```

### 【例4-6】输出一个菱形图

```
1. #include <stdio.h>
2. int main()
3. {
4.     char diamond[][5]={{' ',' ','*'},
5.                          {' ','*',' ','*'},
6.                          {'*',' ',' ',' ','*'},
7.                          {' ','*',' ','*'},
8.                          {' ',' ','*'}};
9.     int i,j;
10.    for (i=0;i<5;i++)
11.    {
12.        for (j=0;j<5;j++)
13.            printf("%c",diamond[i][j]);
14.        printf("\n");
15.    }
16.    return 0;
17.}
```

## 4.4.3 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。

“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。

## 4.4.3 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
---	---	---	---	---	----	----	----	----	----



## 4.4.4 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。

```
1. #include <stdio.h>
2. int main()
3. {
4.     char c[15]={'I',' ','a','m',' ','a',' ','s','t','u','d','e','n','t','.'};
5.
6.     int i;
7.     for(i=0;i<15;i++)
8.         printf("%c",c[i]);
9.     printf("\n");
10.    return 0;
11.}
```

```
#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。



## 4.4.4 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。

```
printf("%o",c);
```

//用八进制形式输出数组c的起始地址

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

//以字符串形式输出字符数组c的内容

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。

## 4.4.5 字符串处理函数

- 输出字符串的函数
- 输入字符串的函数
- 字符串连接函数
- 字符串复制函数
- 字符串比较函数
- 测字符串长度的函数
- 转换为大小写的函数

### 注意

- 字符串处理函数属于**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。

## 4.4.5 字符串处理函数

### ■ 输出字符串的函数

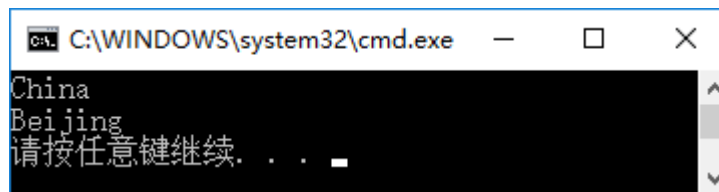
#### puts(字符数组)

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。

```
1. #include <stdio.h>
2. int main()
3. {
4.     char str[]={"China\nBeijing"};
5.     puts(str);
6.     return 0;
7. }
```



## 4.4.5 字符串处理函数

### ■ 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);
```

//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："Computer"外加一个'\0'），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/输入一个字符串。



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

## 4.4.5 字符串处理函数

### ■ 字符串连接函数

`strcat(字符串数组1, 字符串数组2)`

作用：把两个字符串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，结果放在字符串数组1中，函数调用后得到一个函数值——字符串数组1的地址。

字符串数组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));
```

思考：以下代码的输出

```
char str[100] = "China";  
strcat(str, str);  
printf("%s", str);
```

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

思考：对于strcpy而言，通过实验验证下面三种情况时的结果：

- ① `n > strlen(str2);`
- ② `n > strlen(str2);`
- ③ `n == strlen(str2);`

## 4.4.5 字符串处理函数

### ■ 字符串复制函数

**strcpy(字符数组1, 字符串2)**

执行后, str1:

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

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。
- 可以用strncpy函数将字符串str2中前面n个字符复制到字符数组str1中去。



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

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

将str2中最前面n个字符复制到str1中，取代str1中原有的最前面n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'）。



注意

- 对两个字符串比较不能直接用 $\text{str1} > \text{str2}$ 进行比较，因为 $\text{str1}$ 和 $\text{str2}$ 代表地址而不代表数组中全部元素，而只能用 $(\text{strcmp}(\text{str1}, \text{str2}) > 0)$ 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。

## 4.4.5 字符串处理函数

### ■ 字符串比较函数

**strcmp(字符串1, 字符串2)**

- 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。
  - 如全部字符相同，则认为两个字符串相等；
  - 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。
  - 如果字符串1与字符串2相同，则函数值为0。
  - 如果字符串1>字符串2，则函数值为一个正整数。
  - 如果字符串1<字符串2，则函数值为一个负整数。
- 可以用strncmp函数比较字符串1和字符串2的前面n个字符。

strncmp(str1, str2, n);

## 4.4.5 字符串处理函数

### ■ 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)。

```
1. #include <stdio.h>
2. #include <string.h>
3. int main()
4. {
5.     char str[10]="China";
6.     printf("%d,%d\n",strlen(str),sizeof(str));
7. }
```

### ■ 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。

## 4.4.6 字符数组应用举例

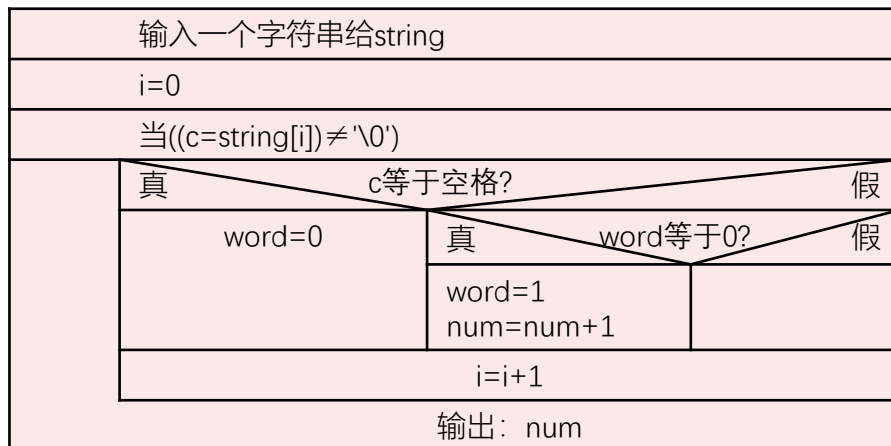
### ■ 【例4-7】输入一行字符，统计其中有多少个单词，单词之间用空格分隔开

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。



```
1. #include <stdio.h>
2. int main()
3. {
4.     char string[81];
5.     int i,num=0,word=0;
6.     char c;
7.     gets(string);    //输入一个字符串给字符数组
8.     for(i=0;(c=string[i])!='\0';i++) //字符'\0'结束循环
9.         if(c==' ') word=0; //若是空格字符，使word置0
10.        else if(word==0)    //若不是空格字符且word原值为0
```

```
11.     {
12.         word=1;    //使word置1
13.         num++;    //num累加1，表示增加一个单词
14.     }
15.     printf("There are %d words in this line.\n",num);
16.     return 0;
17. }
```

## 4.4.6 字符数组应用举例

### ■ 【例4-8】有3个字符串,要求找出其中“最大”者

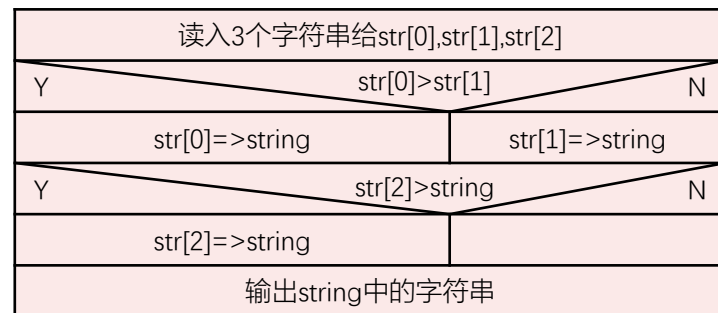
str[0]:	H	o	l	l	a	n	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[1]:	C	h	i	n	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[2]:	A	m	e	r	i	c	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

```
1. #include<stdio.h>
2. #include<string.h>
3. int main()
4. {
5.     char str[3][20];    //定义二维字符数组
6.     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
7.     int i;
8.     for(i=0;i<3;i++) gets(str[i]); //读入3个字符串

9.     if(strcmp(str[0],str[1])>0)    //若str[0]大于str[1]
10.        strcpy(string,str[0]);    //把str[0]的字符串赋给字符数组string
11.     else
12.        strcpy(string,str[1]);    //把str[1]的字符串赋给字符数组string

13.     if(strcmp(str[2],string)>0)    //若str[2]大于string
14.        strcpy(string,str[2]);    //把str[2]的字符串赋给字符数组string

15.     printf("\nthe largest string is:\n%s\n",string); //输出string
16.     return 0;
17. }
```



(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。

## 4.5 枚举类型

**enum [枚举名]{枚举元素列表}**

```
enum Weekday {sun, mon, tue, wed, thu, fri, sat};
```

如果一个变量只有几种可能的值，则可以定义为**枚举**(enumeration)**类型**，所谓“枚举”就是指把可能的值一一列举出来，变量的值只限于列举出来的值的范围内。声明枚举类型用enum开头。花括号中的sun, mon, ..., sat称为**枚举元素**或**枚举常量**。

```
enum Weekday weekday, weekend;
```

枚举类型

枚举变量

也可以不声明有名字的枚举类型，而直接定义枚举变量：

```
enum {sun, mon, tue, wed, thu, fri, sat} weekday, weekend;
```

- (1) C编译对枚举类型的枚举元素按常量处理，故称枚举常量。不要因为它们是标识符(有名字)而把它们看作变量，不能对它们赋值。
- (2) 每一个枚举元素都代表一个整数，C语言编译按定义时的顺序默认它们的值为0,1,2,3,4,5…。也可以在定义枚举类型时显式地指定枚举元素的数值。
- (3) 枚举元素可以用来作判断比较。枚举元素的比较规则是按其在初始化时指定的整数来进行比较的。

## 4.5 枚举类型

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

2. enum Weekday {sun=-1, mon, tue=20, wed, thu, fri, sat};
3. enum Nation {bai, chaoxian, han, tujia, yao}; //按汉语拼音字典序排列
4. int main() {
5.     printf("%lu %lu\n", sizeof(enum Weekday), sizeof(enum Nation));
6.     printf("%lu %lu\n", sizeof(sun), sizeof(bai));
7.     printf("%d %d %d %d %d\n", bai, chaoxian, han, tujia, yao);
8.     printf("%d %d %d %d %d %d %d\n", sun, mon, tue, wed, thu, fri, sat);
9. }
```

```
4 4
4 4
0 1 2 3 4
-1 0 20 21 22 23 24
```

## 4.5 枚举类型

- 【例4-9】口袋中有红、黄、蓝、白、黑5种颜色的球若干个。每次从口袋中先后取出3个球，问得到3种不同颜色球的可能取法

```
1. #include <stdio.h>
2. int main()
3. {
4.     enum Color {red,yellow,blue,white,black};
5.     enum Color i,j,k,pri; //定义枚举变量i,j,k,pri
6.     int n,loop;
7.     n=0;
8.     for(i=red;i<=black;i++) //外循环使i从red变到black
9.         for(j=red;j<=black;j++) //中循环使j从red变到black
10.            if(i!=j) //如果二球不同色
11.            {
12.                for (k=red;k<=black;k++) //内循环使k从red变到black
13.                    if ((k!=i) && (k!=j)) //如果3球不同色
14.                    {
15.                        n=n+1; //符合条件的次数
16.                        printf("%-4d",n); //输出当前是第几个符合条件的组合
```

```
17.         for(loop=1;loop<=3;loop++) //先后对3个球分别处理
18.         {
19.             switch (loop) //loop的值从1变到3
20.             {
21.                 case 1: pri=i;break;
22.                 case 2: pri=j;break;
23.                 case 3: pri=k;break;
24.                 default:break;
25.             }
26.             switch (pri) //根据球的颜色输出相应的文字
27.             {
28.                 case red:printf("%s","red");break;
29.                 case yellow: printf("%s","yellow");break;
30.                 case white: printf("%s","white");break;
31.                 case black: printf("%s","black"); break;
32.                 default:break;
33.             }
34.         }
35.         printf("\n");
36.     }
37. }
38. }
```

## 4.6 结构体

**struct 结构体名**  
**{成员表列};**

C语言允许用户自己建立由不同类型数据组成的组合型的数据结构，它称为**结构体** (structure)。

在程序中建立一个结构体类型：

sno	name	sex	class	nation	age
2021123456	ZhangSan	F	T	2	18

```
enum Nation { bai, chaoxian, han, tujia, ... };
              0      1      2      3

struct Student
{
    int sno;           //学号为整型
    char name[20];     //姓名为字符串
    char sex;          //性别为字符型 (F: 女生, M: 男生)
    char class;        //班级为字符型 (T: 图灵, D: 金科)
    enum Nation nation; //民族为枚举类型
    int age;           //年龄为整型
};                    //注意最后有一个分号
```

结构体类型的名字是由一个关键字**struct**和结构体名组合而成的。结构体名由用户指定，又称“结构体标记”(structure tag)。

花括号内是该结构体所包括的子项，称为结构体的成员(member)。对各成员都应进行类型声明，即 **类型名 成员名;**

“成员列表”(member list)也称为“域表”(field list)，每一个成员是结构体中的一个域。成员名命名规则与变量名相同。



## 4.6 结构体

(1) 结构体类型并非只有一种，而是可以设计出许多种结构体类型，各自包含不同的成员。

(2) 成员可以属于另一个结构体类型。

sno	name	sex	class	nation	age	birthday		
						year	month	day

```
struct Date                                //声明一个结构体类型 struct Date
{
    int year;                             //年
    int month;                            //月
    int day;                              //日
};
```

```
struct Student                             //声明一个结构体类型 struct Student
{
    int sno;
    char name[20];
    char sex;
    char class;
    enum Nation nation;
    int age;
    struct Date birthday;                 //成员birthday属于struct Date类型
};
```

## 4.6.1 定义结构体类型变量

### 1. 先声明结构体类型，再定义该类型的变量

```
struct Student
{
    int sno;           //学号为整型
    char name[20];     //姓名为字符串
    char sex;          //性别为字符型
    char class;        //班级为字符型
    enum Nation nation; //民族为枚举型
    int age;           //年龄为整型
};                    //注意最后有一个分号
```

```
struct Student  student1, student2;
```

结构体类型名      结构体变量名

sutdent1:	2021123456	ZhangSan	F	T	2	18
student2:	2021654321	LiSi	M	D	4	19

### 2. 在声明类型的同时定义变量

```
struct Student
{
    int sno;
    char name[20];
    char sex;
    char class;
    enum Nation nation;
    int age;
} student1, student2;
```

struct 结构体名  
{  
    成员表列  
}变量名表列;

### 3. 不指定类型名而直接定义结构体类型变量

```
struct
{
    成员表列
}变量名表列;
```

## 4.6.1 定义结构体类型变量

- 结构体类型与结构体变量是不同的概念，不要混淆。只能对变量赋值、存取或运算，而不能对一个类型赋值、存取或运算。在编译时，对类型是不分配空间的，只对变量分配空间
- 要结构体类型中的成员名可以与程序中的变量名相同，但二者不代表同一对象
- 对结构体变量中的成员（即“域”），可以单独使用，它的作用与地位相当于普通变量

## 4.6.2 结构体变量的初始化和引用

■ 【例4-9】 把一个学生信息(含学号、姓名、性别、班级、民族)放在一个结构体变量中，然后输出这个学生的信息

```
1. #include <stdio.h>
2. int main()
3. {
4.     enum Nation {bai, chaoxian, han, tujia, yao};
5.     struct Student          //声明结构体类型struct Student
6.     {
7.         int sno;            //以下6行为结构体的成员
8.         char name[20];
9.         char sex;
10.        char class;
11.        enum Nation nation;
12.        int age;
13.    } stu = {2021123456, "ZhangSan", 'F', 'T', han, 18}; //定义结构体变量stu并初始化
14.    printf("NO.:%ld\nname:%s\nsex:%c\nclass:%c\nnation:%d\nage:%d\n",
15.        stu.sno, stu.name, stu.sex, stu.class, stu.nation, stu.age);
16.    return 0;
17. }
```

## 4.6.2 结构体变量的初始化和引用

- (1) 在定义结构体变量时可以对它的成员初始化。初始化列表是用花括号括起来的一些常量，这些常量依次赋给结构体变量中的各成员。

**注意**

对结构体变量初始化，不是对结构体类型初始化

- (2) 可以引用结构体变量中成员的值，引用方式为 **结构体变量名.成员名**

```
student1.sno=10010;
```

/\*已定义了student1为student类型的结构体变量，则student1.sno表示student1变量中的sno成员，即student1的sno(学号)成员\*/

**“.”是成员运算符**，它在所有的运算符中优先级最高，因此可以把student1.sno作为一个整体来看待，相当于一个变量。

```
printf("%s\n",student1); //企图用结构体变量名输出所有成员的值
```



不能企图通过输出结构体变量名来达到输出结构体变量所有成员的值。只能对结构体变量中的各个成员分别进行输入和输出。

## 4.6.2 结构体变量的初始化和引用

- (3) 如果成员本身又属一个结构体类型，则要用若干个成员运算符，一级一级地找到最低的一级的成员。只能对最低级的成员进行赋值或存取以及运算。

```
student1.sno=10010;    //结构体变量student1中的成员sno
student1.birthday.month=6;    //结构体变量student1中的成员birthday中的成员month
```

- (4) 对结构体变量的成员可以像普通变量一样进行各种运算（根据其类型决定可以进行的运算）。

```
student2.class = student1.class;    //赋值运算
student1.age++;    //自加运算
```

- (5) 同类的结构体变量可以互相赋值。

```
student1=student2;    //假设student1和student2已定义为同类型的结构体变量
```

- (6) 可以引用结构体变量成员的地址，也可以引用结构体变量的地址(结构体变量的地址主要用作函数参数，传递结构体变量的地址)。但不能整体读入结构体变量。

```
scanf("%d",&student1.sno);    //输入student1.sno的值
printf("%x",&student1);    //输出结构体变量student1的起始地址
```

- (7) 不能整体读入结构体变量，若要输入整个结构体，只能输入其每个成员



```
scanf("%d,%s,%c,%c,%d\n",&student1);
```



```
scanf("%d,%s,%c,%c,%d\n",&student1.sno, student1.name, &student1.sex,
&student1.class, &student1.nation);
```

## 4.6.3 结构体变量的存储

### ■ 一般规律:

- $\text{sizeof}(\text{对象}) = \text{sizeof}(\text{数据成员1}) + \text{sizeof}(\text{数据成员2}) + \dots + \text{sizeof}(\text{数据成员n})$
- $\text{address}(\text{数据成员i}) = \text{address}(\text{对象}) + \text{sizeof}(\text{数据成员1}) + \text{sizeof}(\text{数据成员2}) + \dots + \text{sizeof}(\text{数据成员i-1})$

### ■ 特殊情况:

- 空类: 类中没有任何数据成员
  - 若对象不占用内存空间, 就无法获得其地址
  - 空类的实际长度为1字节
- 内存对齐
  - 内存对齐可以加速数据成员的访问
  - gcc/g++编译器默认将每个数据成员对齐至其大小的最近整数倍

## 4.6.3 结构体变量的存储

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

2.  enum Nation {bai, chaoxian, han, tujia, yao};

3.  struct Date          //声明一个结构体类型 struct Date
4.  {
5.      int year;
6.      int month;
7.      int day;
8. };

9.  struct Student       //声明结构体类型struct Student
10. {
11.     int sno;
12.     char name[20];
13.     char sex;
14.     char class;
15.     enum Nation nation;
16.     int age;
17.     struct Date birthday;
18. };

19. int main()
20. {
21.     printf("%d %d\n", sizeof(struct Date), sizeof(struct Student));
22. }
```



## 4.7 共用体

使几个不同类型的变量共享同一段内存的结构，称为“共用体”类型的结构。

```
union共用体名 {  
    成员表列  
} 变量表列;
```

1000地址

短整型变量i

字符型变量ch

实型变量f

```
union Data //表示不同类型的变量i,ch,f可以存放到同一段存储单元中  
{  
    int i;  
    char ch;  
    float f;  
} a,b,c; //在声明类型同时定义变量
```

```
union Data //声明共用体类型  
{  
    int i;  
    char ch;  
    float f;  
};  
union Data a,b,c; //用共用体类型定义变量
```

```
union //没有定义共用体类型名  
{  
    int i;  
    char ch;  
    float f;  
}a,b,c;
```

“共用体”与“结构体”的定义形式相似。但它们的含义是不同的。

- 结构体变量所占内存长度是各成员占的内存长度之和。每个成员分别占有其自己的内存单元。
- 而共用体变量所占的内存长度等于最长的成员的长度。几个成员共用一个内存区。

## 4.7.1 共用体类型数据的特点

(1) 同一个内存段可以用来存放几种不同类型的成员，但在每一瞬时只能存放其中一个成员，而不是同时存放几个。

(2) 可以对共用体变量初始化，但初始化表中只能有一个常量。

```
union Data a={1,'a',1.5};
```



(3) 共用体变量中起作用的成员是最后一次被赋值的成员，在对共用体变量中的一个成员赋值后，原有变量存储单元中的值就被取代。

(4) 共用体变量的地址和它的各成员的地址都是同一地址。

(5) 不能对共用体变量名赋值，也不能企图引用变量名来得到一个值，同类型的共用体变量互相赋值。

(6) 只有先定义了共用体变量才能引用它，但应注意，不能引用共用体变量，而只能引用共用体变量中的成员。

```
printf("%d",a);
```



```
printf("%d",a);
```



(7) 共用体类型可以出现在结构体类型定义中，也可以定义

共用体数组。反之，结构体也可以出现在共用体类型定义中，数组也可以作为共用体的成员。

```
union Date
```

```
{
```

```
    int i;
```

```
    char ch;
```

```
    float f;
```

```
} a;
```

```
a.i=97;
```

```
printf("%d",a.i); //输出整数97
```

```
printf("%c",a.ch); //输出字符'a'
```

```
printf("%f",a.f); //输出实数0.000000
```

```
a=1; //不能对共用体变量赋值，赋给谁？
```

```
int m=a; //企图引用共用体变量名以得到一个  
值赋给整型变量m
```



```
b=a; //a和b是同类型的共用体变量，合法
```

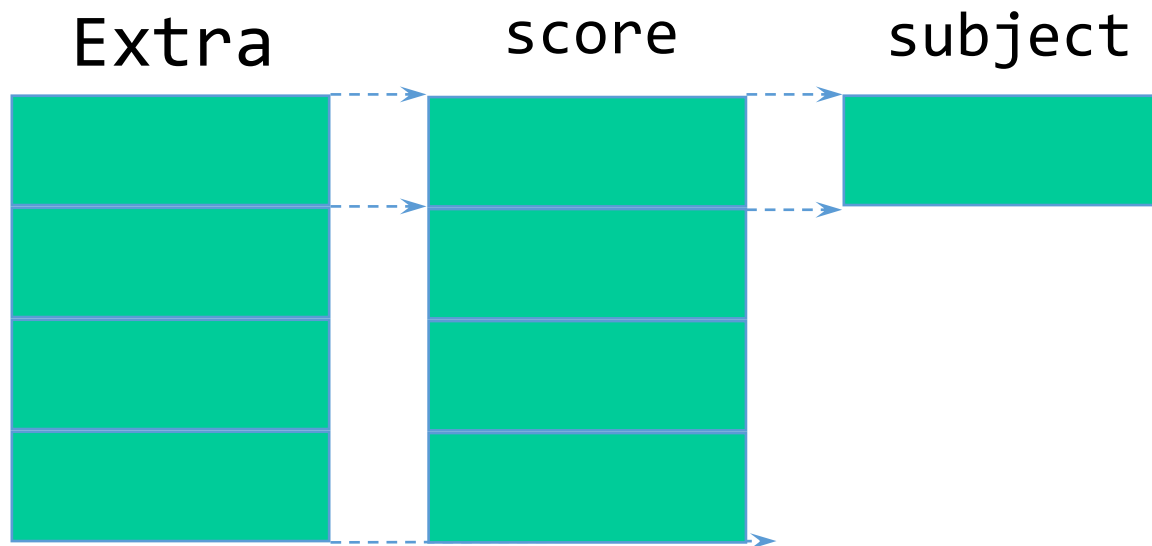


## 4.7.2 共用体变量的存储

```
1. #include <stdio.h>
2. enum Nation {bai, chaoxian, han, tujia, yao};
3. struct Date {int year; int month; int day;};
4. union Extra {
5.     float score;
6.     char subject;
7. };
8. int main() {
9.     printf("%d %d\n", sizeof(struct Extra),
10.           sizeof(struct Student));
11. }
```

```
12. struct Student
13. {
14.     int sno;
15.     char name[20];
16.     char sex;
17.     char class;
18.     enum Nation nation;
19.     int age;
20.     struct Date birthday;
21.     char category;
22.     union Extra extra;
23. }
```

共用体所占的内存空间大小等于其所占空间最大的成员域的大小

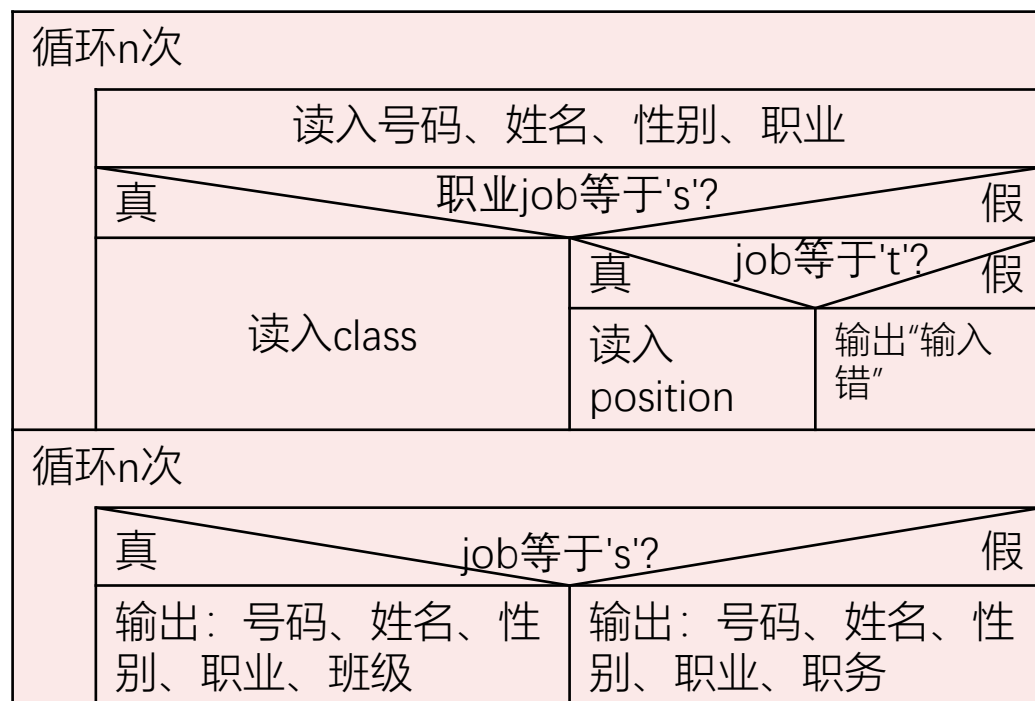


## 4.7.2 共用体应用举例

■ 【例4-10】有若干个人的数据，其中有学生和教师。学生的数据中包括：姓名、号码、性别、职业、班级。教师的数据包括：姓名、号码、性别、职业、职务。要求用同一个表格来处理。

num	name	sex	job	class(班) position(职务)
101	Li	f	s	501
102	Wang	m	t	prof

```
struct                                //声明无名结构体类型
{
    int num;                          //成员num(编号)
    char name[10];                    //成员name(姓名)
    char sex;                         //成员sex(性别)
    char job;                         //成员job(职业)
    union                             //声明无名共用体类型
    {
        int clas;                    //成员clas(班级)
        char position[10];           //成员position(职务)
    }category;                       //成员category是共用体变量
}person[2];                          //定义结构体数组person，有两个元素
```



## 4.7.2 共用体应用举例

```
1. int main()
2. {
3.     int i;
4.     for(i=0;i<2;i++)
5.     {
6.         printf("please enter the data of person:\n");
7.         scanf("%d %s %c %c",&person[i].num,&person[i].name,&person[i].sex,&person[i].job);           //输入前4项
8.         if(person[i].job=='s') scanf("%d",&person[i].category.clas);                           //如是学生, 输入班级
9.         else if(person[i].job=='t') scanf("%s",person[i].category.position);                   //如是教师, 输入职务
10.        else printf("Input error!");                                                         //如job不是's'和't', 显示“输入错误”
11.    }

12.    printf("\n");
13.    printf("No.namesex job class/position\n");
14.    for(i=0;i<2;i++)
15.    {
16.        if (person[i].job=='s') //若是学生
17.            printf("%-6d%-10s%-4c%-4c%-10d\n",person[i].num,person[i].name,person[i].sex,
18.                person[i].job,person[i].category.clas);
19.        else //若是教师
20.            printf("%-6d%-10s%-4c%-4c%-10s\n",person[i].num, person[i].name,person[i].sex,
21.                person[i].job,person[i].category.position);
22.    }

23.    return 0;
24.}
```

# 综合应用：生日祝福

## ■ 定义Student数据结构

- 学号：int sno
- 姓名：char name[20]
- 年龄：int age
- 性别：char sex ('M': 男性, 'F': 女性)
- 班级：char class ('T': 图灵班, 'D': 金科班)
- 民族：enum Nation { bai, chaoxian, han, tujia, yao, ... } nation
- 生日：struct Date { int year; int month; int day; } birthday
- 类别：char category ('N': 统考生; 'B': 保送生)
- 其他：union Extra { float score; char subject; } extra
  - score: 高考成绩
  - subject: 获奖的竞赛学科 ('M': 数学, 'P': 物理, 'I': 信息学, 'C': 化学, 'B': 生物)

```
struct Student
{
    int sno;
    char name[20];
    char sex;
    char class;
    enum Nation nation;
    int age;
    struct Date birthday;
    char category;
    union Extra extra;
}
```

# 综合应用：生日祝福

## ■ 定义学生结构体数组并赋初值

```
struct Student students[] = {  
    {2021123456, "ZhangSan", 'F', 'T', han, 18, {2003, 10, 28}, 'B', 'I'},  
    {2021654321, "LiSi", 'M', 'D', yao, 19, {2002, 11, 22}, 'N', 680}  
};
```

## ■ 查看今天的日期

```
#include <time.h>  
  
struct tm {  
    int tm_sec; //代表目前秒数，正常范围为0-59，但允许至61秒  
    int tm_min; //代表目前分数，范围0-59  
    int tm_hour; //从午夜算起的时数，范围为0-23  
    int tm_mday; //目前月份的日数，范围01-31  
    int tm_mon; //代表目前月份，从一月算起，范围从0-11  
    int tm_year; //从1900 年算起至今的年数  
    int tm_wday; //一星期的日数，从星期一算起，范围为0-6  
    int tm_yday; //从今年1 月1 日算起至今的天数，范围为0-365  
    int tm_isdst; //日光节约时间的旗标  
};  
  
time_t time( time_t * ) ; //返回从1970年1月1日 (MFC是1899年12月31日) 0时0分0秒，到现在的秒数  
struct tm * localtime(const time_t * timer); //将日历时间转为本地时间
```

# 综合应用：生日祝福

```
1. #include <stdio.h>
2. #include <time.h>

3. enum Nation {bai, chaoxian, han, tujia, yao};
4. struct Date {
5.     int year;
6.     int month;
7.     int day;
8. };

9. union Extra {
10.     float score;
11.     char subject;
12. };

13. struct Student
14. {
15.     int sno;
16.     char name[20];
17.     char sex;
18.     char class;
19.     enum Nation nation;
20.     int age;
21.     struct Date birthday;
22.     char category;
23.     union Extra extra;
24. };
```

```
25. int main() {
26.     time_t rawtime;
27.     struct tm *target_time;
28.     int this_year, this_month, this_day, i;

29.     struct Student students[] = {
30.         {2021123456, "ZhangSan", 'F', 'T', han, 18,
31.          {2003, 10, 28}, 'B', 'I'},
32.         {2021654321, "LiSi", 'M', 'D', yao, 19,
33.          {2002, 11, 22}, 'N', 680}
34.     };

35.     time(&rawtime);
36.     target_time = localtime(&rawtime);
37.     this_year = target_time->tm_year + 1900;
38.     this_month = target_time->tm_mon + 1;
39.     this_day = target_time->tm_mday;
40.     printf("Today is: %d-%d-%d\n",
41.            this_year, this_month, this_day);

42.     for (i=0; i<sizeof(students)/sizeof(struct Student); i++)
43.         if (students[i].birthday.month == this_month &&
44.             students[i].birthday.day == this_day) {
45.             printf("\t%s(%d)'s %d birthdy.\n",
46.                    students[i].name, students[i].sno, students[i].age);
47.         }
```



## 4.8 位域

### ■ 位域的声明形式

- 数据类型说明符 成员名 : 位数;

### ■ 位域的作用

- 通过“打包”，使类的不同成员共享相同的字节，从而节省存储空间。

### ■ 注意事项

- 具体的打包方式，因编译器而异；
- 只有bool、char、int和枚举类型的成员，允许定义为位域；
- 节省空间，但可能增加时间开销

# 示例：位域存储学生的成绩信息

```
1. struct Student
2. {
3.     int sno;
4.     char name[20];
5.     char sex;
6.     char class;
7.     enum Nation nation;
8.     int age;
9.     struct Date birthday;
10.    char category;
11.    union Extra extra;
12.};
```

sizeof(struct Student) == 56

```
1. struct Student_
2. {
3.     int sno;
4.     char name[20];
5.     char sex:1;
6.     char class:1;
7.     char category:1;
8.     enum Nation nation;
9.     int age;
10.    struct Date birthday;
11.    union Extra extra;
12.};
```

sizeof(struct Student\_) == 52

位运		含义
&		按位与
		按位或
^		按位异或
~		取反
<<		左移
>>		右移

说明：

1. 位运算符中除 ~ 以外，均为二目（元）运算符，即要求两侧各有一个运算量。

2. 运算量只能是整型或字符型的数据，不能为实型数据。

下面对各运算符分别介绍如下：

一、“按位与”运算符（&）

参加运算的两个运算量，如果两个相应的位都为 1，则该位的结果值为 1，否则为 0。即：

$0 \& 0 = 0$ ； $0 \& 1 = 0$ ； $1 \& 0 = 0$ ； $1 \& 1 = 1$ ；

例如：3 & 5 并不等于不等于 8，这是按位与。先把 3 和 5 以补码表示，再进行按位与运算。

3 的补码：0 0 0 0 0 0 1 1

5 的补码：0 0 0 0 0 1 0 1

-----

&：0 0 0 0 0 0 0 1

它是 1 的补码。因此，3 & 5 的值得 1。

按位与有一些特殊的用途：

1. 清零。如果想将一个单元清零，即使其全部二进制位为0，只要找一个数，它的补码形式中各位的值符合以下条件：原来的数中为1的位，新数中相应位为0。然后使二者进行&

运算。

如：原有数为 0 0 1 0 1 0 1 1，另找一个数，设它为 1 0 0 1 0 1 0 0，它符合以上条件，即在原数为 1 的位置上，它的位值均为 0。将两个数进行 & 运算：

$$\begin{array}{r} 00101011 \\ \& 10010100 \\ \hline 00000000 \end{array}$$

其道理是显然的。当然也可以不用 10010100 这个数而用其它数（如 0 1 0 0 0 1 0 0 也可以，只要符合上述条件即可。

2.取一个数中某些指定位。如一个整数 a (2 个字节), 如只想要其中的低字节。只需将 a 与  $(377)_8$  按位与即可。见图 11.2。

```

-----
a | 00 10 11 00 | 10 10 11 00 |
-----
-----
b | 00 00 00 00 | 11 11 11 11 |
-----
-----
c | 00 00 00 00 | 10 10 11 00 |
-----

```

图11.2

$c = a \& b$ ,  $b$  为八进制的 377,  $c$  只取  $a$  的低字节, 高字节为 0。

如果想取两个字节中的高字节，只需： $c = a \& 0177400$ （ $0177400$ 表示八进制的  
 $177400$ ）。见图 11.3。

```
-----  
a | 00 10 11 00 | 10 10 11 00 |  
-----  
-----  
b | 11 11 11 11 | 00 00 00 00 |  
-----  
-----  
c | 00 10 11 00 | 00 00 00 00 |  
-----
```

图 11.3



3.要想将哪一位保留下来，就与一个数进行&运算，此数在该位取1，如：有一数  
0 1 0 1 0 1 0 0,想把其中左面第3、4、5、7、8位保留下来，  
可以这样

$$\begin{array}{rcl}
 & 01010100 & \text{(十进制数84)} \\
 (& \&) & 00111011 & \text{(十进制数59)} \\
 \hline
 & 00010000 & \text{(十进制数16)} \\
 & \text{----} & 
 \end{array}$$

即  $a = 84$  ,  $b = 59$  ,  $c = a \& b = 16$ 。

## 二、按位或运算符 (|)

两个相应位中只要有一个为1，该位的结果值为1。即：0 |  
0 = 0 ; 0 | 1 = 1 ;

$1 \mid 0$   
 $= 1$  , ;  $1 \mid 1 = 1$  。例如  
 $0 \ 6 \ 0 \mid 0 \ 1 \ 7$

将八进制数 6 0 与八进制数 1 7 进行按位或运算。

```

      00110000
(|)   00001111
-----
      00111111
      -----

```

把低 4 位全置 1。如果想使一个数 a 的低 4 位改为 1，只需将，与 0 i 7 进行按位或运算可。

按位或运算常用来对一个数据的某些位定值为 1。如 a 是一个整数 ( i 6 位)，有表达式

a | 0 3 7 7

则低 8 位全置为 1。高 8 位保留原样。

三、“异或”运算符 ( ^ )，也称 X O R 运算符

它的规则是：参加运算的两个相应位同号，则结果为 0（假）；异号则为 1（真）。即：

$0 \wedge 0 = 0$ ；  $0 \wedge 1 = 1$ ；  $1 \wedge 0 = 1$ ；  $1 \wedge 1 = 0$ ； 如：

00111001	(十进制数 57, 八进制数 0 7 1)
^ 00101010	(十进制数 42, 八进制数 0 5 2)
-----	
00010011	(十进制数 19, 八进制数 0 2 3)

即  $071 \wedge 052$ ，结果为  $023$ （八进制数）。

"异或"的意思是：判断两个相应的位值是否为"异"，为"异"（值不同）就取真（1），否则为假（0）。

下面举例说明  $\wedge$  运算符的应用：

（1）使特定位翻转

假设有  $01111010$ ，想使其低4位翻转，即：变为  $0, 0$  变为  $1$ 。可以将它与  $00001111$  进行  $\wedge$  运算，即

$$\begin{array}{r} \phantom{\wedge} \phantom{0000} 01111010 \\ \wedge \phantom{0000} 00001111 \\ \hline \phantom{0000} 01110101 \\ \phantom{0000} \phantom{0000} \phantom{0000} \phantom{0000} \end{array}$$

结果值的低 4 位正好是原数低 4 位的翻转。要使哪几位翻转就将其与 1 进行 ^ 运算的数中该位位置为 1 即可。这是因为原数中值为 1 的位与 1 进行 ^ 运算得 0，原数中的位值 0 与 1 进行 ^ 运算的结果得 1。

(2) 与 0 相 ^，保留原值

如  $012 \wedge 00 = 012$

$$\begin{array}{r} \phantom{0000}01010 \\ \wedge \phantom{0000}00000000 \\ \hline 00001010 \end{array}$$

因为原数中的 1 与 0 进行 ^ 运算得 1,  $0 \wedge 0$  得 0, 故保留原数。

( 3 ) 交换两个值, 不用临时变量

假如  $a = 3$ ,  $b = 4$ 。想将  $a$  和  $b$  的值互换, 可以用以下赋值语句实现:

$a = a \wedge b;$

$b = b \wedge a;$

$a = a \wedge b;$

可以用下面的竖式来说明:

a= 011  
( ^ ) b= 100

-----

a= 111 ----(a ^ b的结果, b已变成 7 )  
( ^ ) b= 100

-----

b= 011 ----(b ^ a的结果, b已变成 3 )  
( ^ ) a= 111

-----

a= 100 ---(a ^ b的结果, a以变成倍 4 )