



中國人民大學

RENMIN UNIVERSITY OF CHINA

信息学院

SCHOOL OF INFORMATION

程序设计荣誉课程

6. 数据结构1——数组与自定义数据类型

授课教师：游伟 副教授、孙亚辉 副教授

授课时间：周二14:00 – 15:30, 周四10:00 – 11:30 (教学三楼3304)

上机时间：周二18:00 – 21:00 (理工配楼二层机房)

课程主页：<https://www.youwei.site/course/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)
逗号运算符:	,
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:	* &

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. 多维数组
3. 字符数组
4. 枚举类型
5. 结构体与共同体
6. 位域与位运算

6.1 一维数组

- 要向计算机输入全班50个学生一门课程的成绩

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



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。

6.1.1 定义一维数组

类型说明符 数组名[常量表达式]

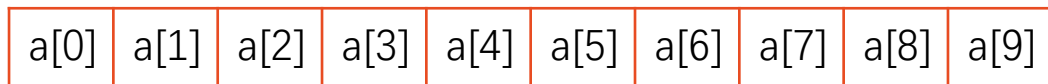
```
int a[10];
```

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。



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]

6.1.2 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
```

6.1.3 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。

6.1.4 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[1];
10.    scanf("%d", &l);
11.    int arr5[l];
12.    printf("%d %d %d %d %d\n",
13.        sizeof(arr1), sizeof(arr2),
14.        sizeof(arr3), sizeof(arr4), sizeof(arr5));
15.}
```

注意：定义数组时，数组大小尽量用常量表达式，不要用变量表达式！如用变量表达式，要确保在数组定义前已对变量进行了正确的赋值。

输入：

5 5

输出：

20 20 20 0 20

6.1.5 一维数组应用举例

- 【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

```
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.}
```



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。

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

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
```

6.1.5 一维数组应用举例

■ 【例6-2】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. . .
```

6.2 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。

6.2.1 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

数组第二维有6个元素

数组第一维有3个元素



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

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]

6.2.2 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）。

6.2.3 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};
```

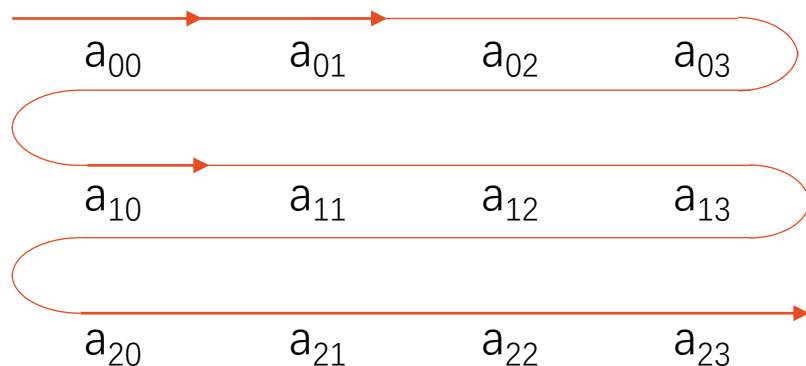
在定义时也可以只对部分元素赋初值而省略第1维的长度，但应分行赋初值。

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

6.2.4 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	$a[2][3]$

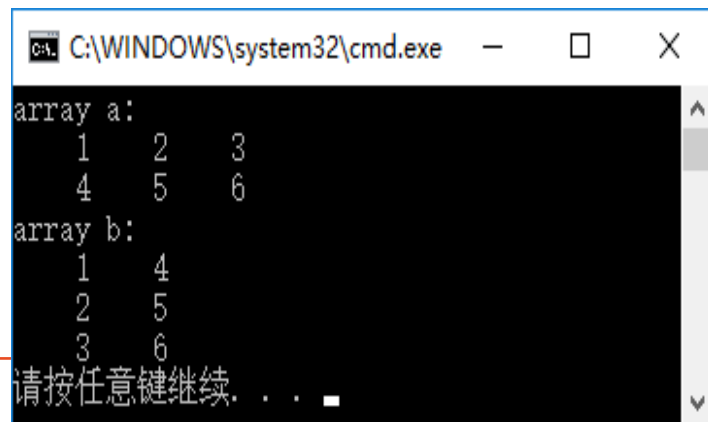
6.2.5 二维数组应用举例

- 【例6-3】 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. }
```



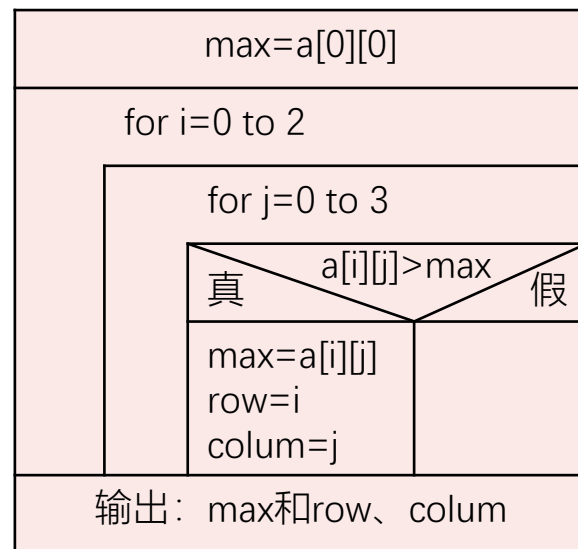
```
C:\WINDOWS\system32\cmd.exe
array a:
 1  2  3
 4  5  6
array b:
 1  4
 2  5
 3  6
请按任意键继续...
```

6.2.5 二维数组应用举例

- 【例6-4】有一个 3×4 的矩阵，要求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。



一维数组与二维数组下标的转换

- 一维数组下标为index; 二维数组下标为[row, col], 长度为M*N
 - 从0开始编号

	第0列	第1列	第2列
第0行	元素0	元素1	元素2
第1行	元素3	元素4	元素5
第2行	元素6	元素7	元素8

- 一维数组下标 → 二维数组下标
 $row = index/N, col = index \% N$
- 二维数组下标 → 一维数组下标
 $index = row * N + col$

- 从1开始编号

	第1列	第2列	第3列
第1行	元素1	元素2	元素3
第2行	元素4	元素5	元素6
第3行	元素7	元素8	元素9

- 一维数组下标 → 二维数组下标
 $row = (index-1)/N+1, col = (index-1)\%N+1$
- 二维数组下标 → 一维数组下标
 $index = (row-1)*N+col$

6.3 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]

6.4 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'?

6.4.1 字符数组的初始化

```
char c[10]={'l',' ','a','m',' ','h','a','p','p','y'};  
char b[10] = "I am happy";  
char a[11] = "I am happy";
```

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]={{ ' ',' ','*'},{' ','*',' ','*'},{'*',' ',' ',' ','*'},{' ','*',' ',' ','*'},{' ',' ',' ','*'}};
```

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

6.4.2 引用字符数组中的元素

【例6-5】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6-6】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.}
```

6.4.3 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。

```
1. #include <stdio.h>
2. #include <string.h>
3. int main()
4. {
5.     char str[10]="China";
6.     printf("%d,%d\n",
7.         strlen(str),
8.         sizeof(str));
9. }
```

5, 10

6.4.3 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
---	---	---	---	---	----	----	----	----	----

6.4.4 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。

6.4.4 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。

6.4.5 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。

6.4.5 字符串处理函数

■ 输出字符串的函数

puts(字符数组)

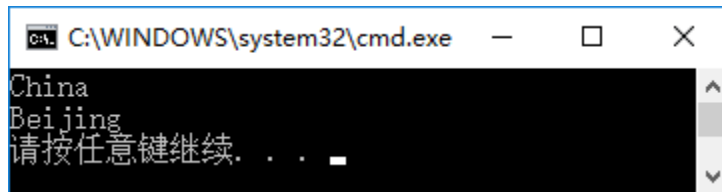
作用：将一个字符串(以'\0'结束的字符序列)输出到终端。

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。

```
/* 模拟实现puts */  
void my_puts(char str[]) {  
    int i = 0;  
    while (str[i] != '\0') {  
        printf("%c", str[i]);  
        i++;  
    }  
    printf("\n");  
}
```

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



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

6.4.5 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。

```
/* 模拟实现gets */
char* my_gets(char str[]) {
    int i = 0; char ch;
    do {
        scanf("%c", &ch);
        if (ch != '\n') str[i++] = ch;
    } while (ch != '\n');
    str[i] = '\0';
    return str;
}
```

```
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);
```


思考：对于strncpy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)` ;

6.4.5 字符串处理函数

■ 字符串复制函数

strcpy(字符数组1, 字符串2)

```
/* 模拟实现strcpy */
char* my_strcat(char str1[], char str2[]) {
    int idx = 0;
    while (str2[idx] != '\0') {
        str1[idx] = str2[idx]; idx++;
    }
    str1[idx] = '\0';
    return str1;
}
```

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

- 作用：将字符串2复制到字符数组1中去。 执行后，str1:

C	h	i	n	a	\0	\0	\0	\0	\0
---	---	---	---	---	----	----	----	----	----
- 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'）。

6.4.5 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);
```

```
/* 模拟实现strcmp */  
int my_strcmp(char str1[], char str2[]) {  
    int idx = 0;  
    while (str1[idx] == str2[idx] &&  
           str1[idx] != '\0' &&  
           str2[idx] != '\0')  
        idx++;  
    return str1[idx] - str2[idx];  
}
```

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。

6.4.5 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)。

■ 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。

```
/* 模拟实现strlen */
unsigned int my_strlen(char str[]) {
    unsigned int len = 0;
    while (str[len] != '\0') len++;
    return len;
}
```

```
/* 模拟实现strlwr */
char* my_strlwr(char str[]) {
    int idx;
    while (str[idx] != '\0') {
        str[idx] = (str[idx]>='A' && str[idx]<='Z' ? str[idx]-'A'+ 'a' : str[idx]);
        idx++;
    }
    return str;
}
```

```
/* 模拟实现strupr */
char* my_strupr(char str[]) {
    int idx;
    while (str[idx] != '\0') {
        str[idx] = (str[idx]>='a' && str[idx]<='z' ? str[idx]-'a'+ 'A' : str[idx]);
        idx++;
    }
    return str;
}
```

6.4.6 字符数组应用举例

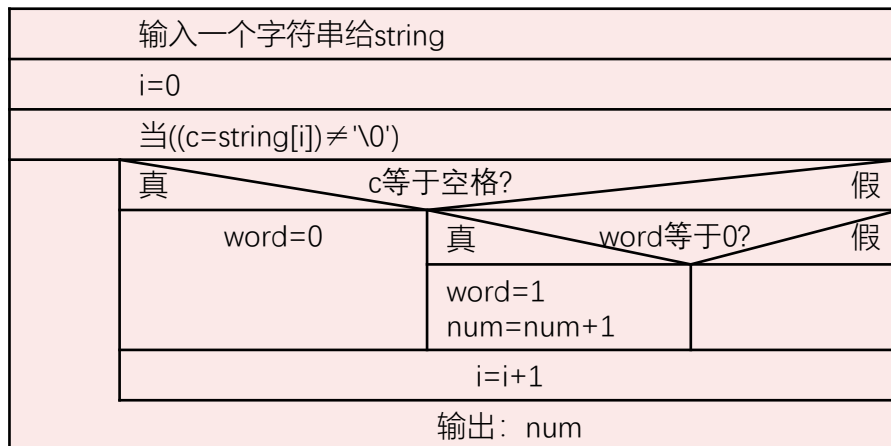
■ 【例6-7】 输入一行字符，统计其中有多少个单词，单词之间用空格分隔开

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. }
```

6.4.6 字符数组应用举例

■ 【例6-8】有3个字符串,要求找出其中“最大”者

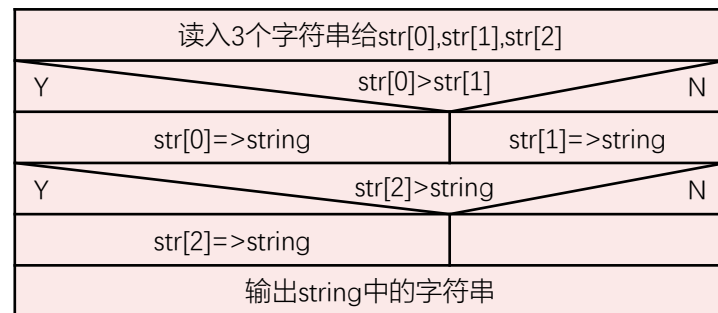
str[0]:	H	o	l	l	a	n	d	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[1]:	C	h	i	n	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0
str[2]:	A	m	e	r	i	c	a	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0	\0

```
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。

思考：对于strncpy而言，通过实验验证下面三种情况时的结果：

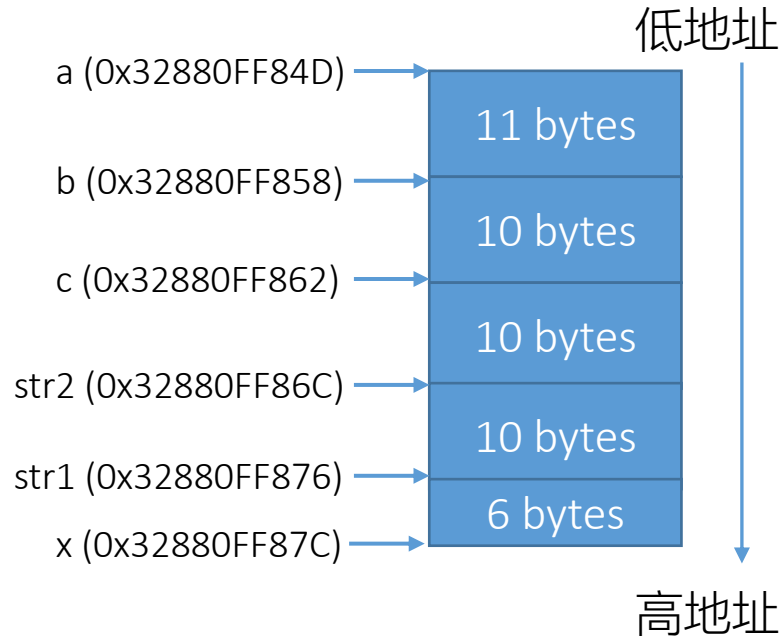
①. `n<strlen(str1)`； ②. `n>strlen(str1)`； ③. `n==strlen(str1)`；

字符数组/字符串思考题验证

思考：如何验证下列代码会不会在
字符数组结尾加一个'\0'？

```
char c[10]={'l',' ','a','m',' ','h','a','p','p','y'};  
char b[10] = "I am happy";  
char a[11] = "I am happy";
```

```
1. #include <stdio.h>  
2. #include <string.h>  
3. int main()  
4. {  
5.     int x;  
6.     char str1[] = "china";  
7.     char str2[10] =  
8.     {'1', '1', '1', '1', '1',  
9.     '1', '1', '1', '1', '\0'}; str2 (0x32880FF86C)  
10.    strncpy(str2, str1, 2);  
11.    printf("%s\n", str2);  
12.    strncpy(str2, str1, 5);  
13.    printf("%s\n", str2);  
14.    strncpy(str2, str1, 6);  
15.    printf("%s\n", str2);  
16.    strncpy(str2, str1, 10);  
17.    printf("%s\n", str2);  
18.    //  
19.    char c[10] = {'I', ' ', 'a', 'm', ' ', 'h', 'a', 'p', 'p', 'y'};  
20.    char b[10] = "I am happy";  
21.    char a[11] = "I am happy";  
22.    printf("%p\n%p\n%p\n%p\n%p\n%p\n%p\n", &x, str1, str2, c, b, a);  
23.    printf("%s\n%s\n%s\n", c, b, a);  
24.    return 0;  
25.}
```



思考：假如从低地址到高地址
分别排布数组c、数组b、数组a，
是否还可以这么验证？

```
ch11111111  
china11111  
china  
china  
00000032880FF87C  
00000032880FF876  
00000032880FF86C  
00000032880FF862  
00000032880FF858  
00000032880FF84D  
I am happychina  
I am happyI am happychina  
I am happy
```

6.5 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 workday, 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} workday, 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。

6.5 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
```


6.5 枚举类型

- 【例6-9】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. }
```

6.6 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
    short age;       //年龄为短整型
};                  //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。

6.6 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;
    short age;
    struct Date birthday; //成员birthday属于struct Date类型
};
```

6.6.1 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
    short age;        //年龄为短整型
};                    //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;
    short 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;
```

6.6.1 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
- 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

6.6.2 结构体变量的初始化和引用

- 【例6-10】 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.        short 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. }
```

6.6.2 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。

6.6.2 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);
```


6.6.3 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
- 编译器默认将每个数据成员对齐至其大小的最近整数倍
- 整个结构体的大小必须是结构体中占用内存空间最大的基础类型成员域大小的整数倍

6.6.3 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.     short age;
17.     struct Date birthday;
18. };
19. int main()
20. {
21.     printf("%d %d\n", sizeof(struct Date), sizeof(struct Student));
22. }
```

6.6.3 结构体变量的存储

■ 如何排布结构体成员域，使得结构体所占内存最小？

```
1. enum Nation {bai, chaoxian, han, tujia, yao};
2. struct Date {int year; int month; int day;};
3. struct Student1
4. {
5.     int sno;
6.     char name[20];
7.     char class;
8.     enum Nation nation;
9.     short age;
10.    struct Date birthday;
11.    char sex;
12.};
```

enum需要
4字节对齐

int需要
4字节对齐

0 → sno
4 →
24 →
25 →
28 → nation
32 → age
34 →
36 → birthday
48 →
49 → sex
52 →

整个结构体中占用字符数最多的基本类型占4个字节，因此结构体大小应是4的整倍数。

`sizeof(struct Student1) == 52`

```
1. enum Nation {bai, chaoxian, han, tujia, yao};
2. struct Date {int year; int month; int day;};
3. struct Student2
4. {
5.     struct Date birthday;
6.     enum Nation nation;
7.     int sno;
8.     short age;
9.     char name[20];
10.    char class;
11.    char sex;
12.};
```

0 → birthday
12 → nation
16 → sno
20 → age
22 →
42 → name
43 → class
44 →

`sizeof(struct Student2) == 44`

方法：把占用内存空间大的成员域放前面，占用内存空间小的成员域放后面

6.7 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  
{  
    short 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  
{  
    short 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  
{  
    short 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。

6.7.1 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
```



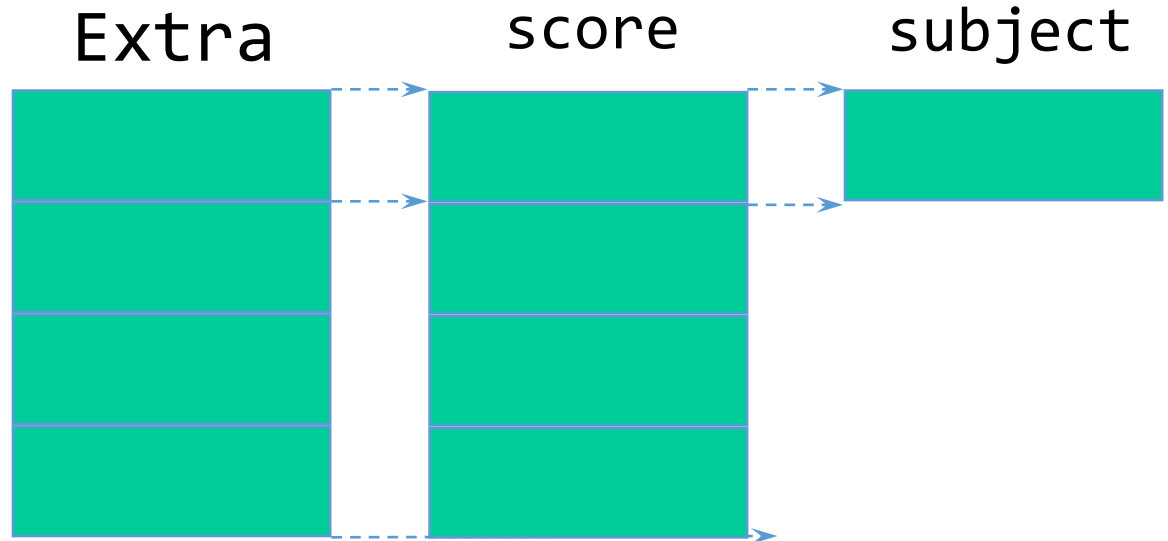
6.7.2 共用体变量的存储

```
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(union 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



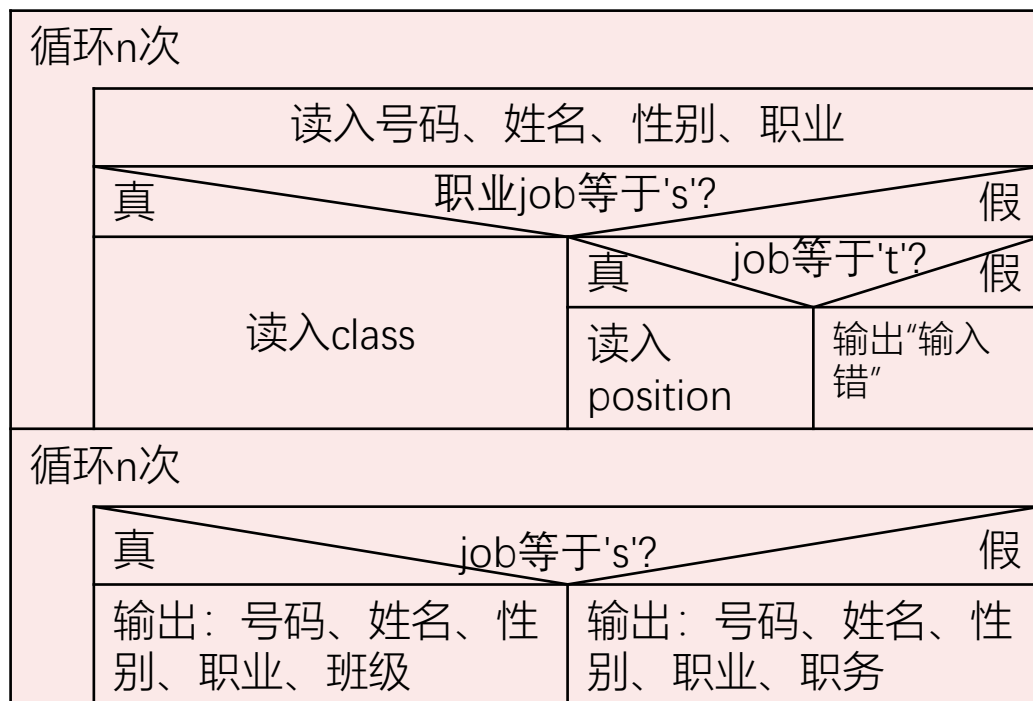
6.7.2 共用体应用举例

- 【例6-11】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
    
```



6.7.2 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.}
```


用typedef声明新类型名

1. 简单用一个新的类型名代替原有类型名

```
typedef int Integer; //指定用Integer为类型名, 作用与int相同  
typedef float Real; //指定用Real为类型名, 作用与float相同
```

2. 命名一个简单的类型名代替复杂的类型表示方法

① 命名一个新类型名代表结构体类型

② 命名一个新类型名代表数组类型

③ 命名一个新类型名代表指针类型

④ 命名一个新类型名代表指向函数的指针类型

```
typedef struct  
{  
    int month;  
    int day;  
    int year;  
}Date; //声明了一个新类型名Date, 代表结构体类型  
Date birthday; //定义结构体类型变量birthday, 不要写成struct Date birthday;  
Date*p; //定义结构体指针变量p, 指向此结构体类型数据 ①
```

```
typedef int Num[100]; //声明Num为整型数组类型名  
Num a; //定义a为整型数组名, 它有100个元素 ②
```

```
typedef char*String; //声明String为字符指针类型  
String p,s[10]; //定义p为字符指针变量, s为字符指针数组 ③
```

```
typedef int (*Pointer)(); //声明Pointer为指向函数的指针类型, 该函数返回整型值  
Pointer p1,p2; //p1,p2为Pointer类型的指针变量 ④
```

用typedef声明新类型名

声明一个新的类型名的方法是：

- ① 先按定义变量的方法写出定义体（如：`int i;`）
- ② 将变量名换成新类型名（例如：将*i*换成Count）
- ③ 在最前面加typedef（例如：`typedef int Count`）
- ④ 然后可以用新类型名去定义变量

简单地说，就是按定义变量的方式把变量名换上新类型名，并在最前面加typedef，就声明了新类型名代表原来的类型。

用typedef声明新类型名

以定义数组类型为例：

- ① 先按定义数组变量形式书写：`int a[100]`
- ② 将变量名a换成自己命名的类型名：`int Num[100]`
- ③ 在前面加上typedef，得到typedef int Num[100]
- ④ 用来定义变量：`Num a`；相当于定义了：`int a[100]`；

以定义字符指针类型为例：

- ① `char *p;` //定义变量p的方式
- ② `char *String;` //用新类型名String取代变量名p
- ③ `typedef char *String;` //加typedef
- ④ `String p;` //用新类型名String定义变量，相当char *p;

习惯上，常把用typedef声明的类型名的第1个字母大写，以便与系统提供的标准类型标识符相区别。

用typedef声明新类型名

- (1) typedef的方法实际上是为特定的类型指定了一个同义字(synonyms)。
 - (2) 用typedef只是对已经存在的类型指定一个新的类型名，而没有创造新的类型。
 - (3) 用typedef声明数组类型、指针类型，结构体类型、共用体类型、枚举类型等，使得编程更加方便。
 - (4) typedef与#define表面实质不同的。#define是在预编译时处理的，它只能作简单的字符串替换，而typedef是在编译阶段处理的，且并非简单的字符串替换。
 - (5) 当不同源文件中用到同一类型数据（尤其是像数组、指针、结构体、共用体等类型数据）时，常用typedef声明一些数据类型。可以把所有的typedef名称声明单独放在一个头文件中，然后在需要用到它们的文件中用#include指令把它们包含到文件中。这样编程者就不需要在各文件中自己定义typedef名称了。
 - (6) 使用typedef名称有利于程序的通用与移植。有时程序会依赖于硬件特性，用typedef类型就便于移植。
-

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)

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

综合应用：生日祝福

■ 定义学生结构体数组并赋初值

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

学习更多：

■ 查看日期

<https://www.runoob.com/cprogramming/c-standard-library-time-h.html>

```
#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
};

typedef long long int time_t; //time_t是long long int类型的别名
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
```

`struct tm *localtime(const time_t *timer)`

timer 的值被分解为 tm 结构，并用本地时区表示。

`time_t time(time_t *timer)`

计算当前日历时间，并把它编码成 time_t 格式。

综合应用：生日祝福

```
1. #include <stdio.h>
2. #include <time.h>
3. typedef enum {
4.     bai, chaoxian, han, tujia, yao
5. } Nation;
6. typedef struct {
7.     int year; int month; int day;
8. } Date;
9. typedef union {
10.     float score; char subject;
11. } Extra;
12. typedef struct
13. {
14.     int sno;
15.     char name[20];
16.     char sex;
17.     char class;
18.     Nation nation;
19.     int age;
20.     Date birthday;
21.     char category;
22.     Extra extra;
23. } Student;
```

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

6.8 位域与位运算

- C语言不仅具备高级语言的功能，也具有低级语言的特点
- 高级语言中数据处理的最小单位一般是字节（Byte），而低级语言可以对每一个二进制位（Bit）进行处理
- C语言提供了对二进制位的处理机制
 - 可以自由地将若干位组合成有意义的信息——位域
 - 可以将一个二进制位作为一个逻辑值进行运算——位运算

6.8.1 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

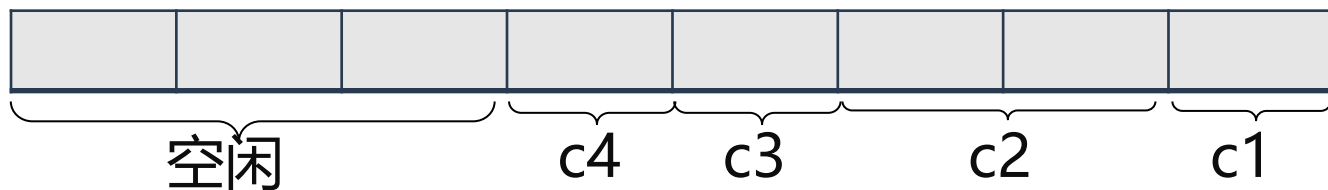
- 节省空间，但可能增加时间开销
- 具体的打包方式，因编译器而异；
- 只有**整数类型**（包括char）和**枚举类型**的成员，允许定义为位域；
- 位域指定的位数必须小于等于该类型所占的位数；
- 由于多个位域可能存储在同一个存储单元中，所以不可以对位域执行取地址操作，否则会出现编译错误；
- 引用位段时，系统自动将它转换成整型；
- 位段可以用%d、%u、%x、%o和%c的格式输出

6.8.1 位域

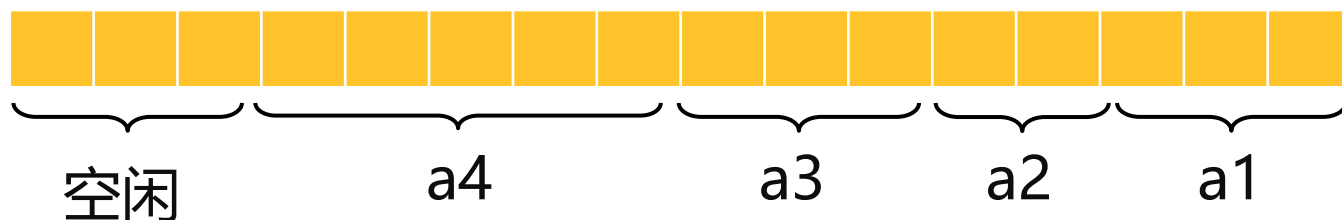
■ 位域在内存中的存储

```
struct sample {  
    int no;  
    char c1:1;  
    char c2:2;  
    char c3:1;  
    char c4:1;  
    short a1:3;  
    short a2:2;  
    short a3:3;  
    short a4:5;  
    short a5:5;  
};
```

相同类型的位段可以在同一块空间中



一个位段必须存储在一个存储单元中，不能跨两个存储单元。



```
1. struct sample_a {
2.     char f1 : 1;
3.     char f2 : 4;
4.     char f3 : 1;
5. } sa1, sa2;

6. struct sample_b {
7.     short f1 : 1;
8.     short f2 : 4;
9.     short f3 : 1;
10.} sb1, sb2;

11.struct sample_c {
12.     char f1 : 1;
13.     short f2 : 4;
14.     int f3 : 1;
15.} sc1, sc2;
```

```
16.struct sample_d {
17.     char f1 : 1;
18.     short f2;
19.     char f3 : 1;
20.} sd1, sd2;

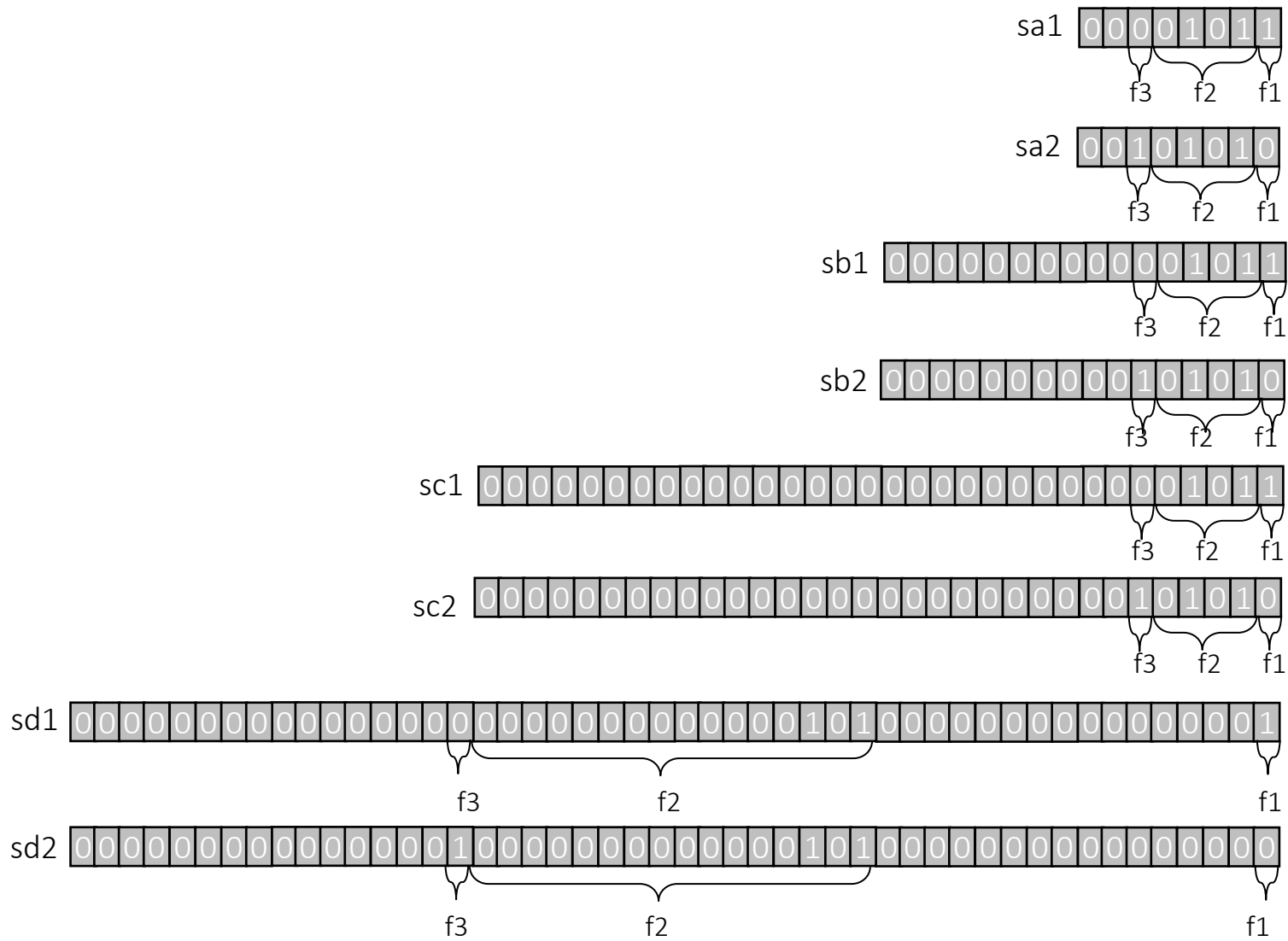
21.struct sample_e {
22.     char f1 : 1;
23.     short f2;
24.     int f3;
25.     char f4 : 1;
26.} se;

27.struct sample_f {
28.     char f1 : 1;
29.     char f4 : 1;
30.     short f2;
31.     int f3;
32.} sf;
```

```
33.int main() {
34.     sa1.f1 = 1; sa1.f2 = 5; sa1.f3 = 0;
35.     printf("%d %x\n", sizeof(sa1), sa1);
36.     sa2.f3 = 1; sa2.f2 = 5; sa2.f1 = 0;
37.     printf("%d %x\n", sizeof(sa2), sa2);
38.     sb1.f1 = 1; sb1.f2 = 5; sb1.f3 = 0;
39.     printf("%d %x\n", sizeof(sa1), sa1);
40.     sb2.f3 = 1; sb2.f2 = 5; sb2.f1 = 0;
41.     printf("%d %x\n", sizeof(sa2), sa2);
42.     sc1.f1 = 1; sc1.f2 = 5; sc1.f3 = 0;
43.     printf("%d %lx\n", sizeof(sc1), sc1);
44.     sc2.f3 = 1; sc2.f2 = 5; sc2.f1 = 0;
45.     printf("%d %lx\n", sizeof(sc2), sc2);
46.     sd1.f1 = 1; sd1.f2 = 5; sd1.f3 = 0;
47.     printf("%d %lx\n", sizeof(sd1), sd1);
48.     sd2.f3 = 1; sd2.f2 = 5; sd2.f1 = 0;
49.     printf("%d %lx\n", sizeof(sd2), sd2);
50.     printf("%d %d\n", sizeof(se), sizeof(sf));
51.     return 0;
52.}
```

结果:

```
1 b
1 2a
2 b
2 2a
4 b
4 2a
6 50001
6 100050000
12 8
```



高地址 ←

→ 低地址

思考：引入位域之后，将占用内存大的域排前面，小的排后面是否还一定能使得整个结构体占用的内存最小？若不能，可否设计一个反例？

6.8.1 位域

■ 示例：位域存储学生的成绩信息

```
1. enum Nation {bai, chaoxian, han, tujia, yao};
2. struct Date {int year; int month; int day;};
3. union Extra { float score; char subject; };
```

```
1. struct Student3
2. {
3.     struct Date birthday;
4.     union Extra extra;
5.     enum Nation nation;
6.     int sno;
7.     short age;
8.     char name[20];
9.     char sex;
10.    char class;
11.    char category;
12.};
```

sizeof(struct Student3) == 52

```
1. struct Student4
2. {
3.     struct Date birthday;
4.     union Extra extra;
5.     enum Nation nation;
6.     int sno;
7.     short age;
8.     char name[20];
9.     char sex : 1;
10.    char class : 1;
11.    char category : 1;
12.};
```

sizeof(struct Student4) == 48

```
1. struct Student5
2. {
3.     char sex : 1;
4.     struct Date birthday;
5.     union Extra extra;
6.     enum Nation nation;
7.     int sno;
8.     char class : 1;
9.     short age;
10.    char category : 1;
11.    char name[20];
12.};
```

sizeof(struct Student5) == 56

6.8.2 位运算

■ 位运算符

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

■ 注意事项

- 参加位运算的数据类型只能是整型或字符型
- 取反是一元运算，其他都是二元运算
- 负数按补码形式参加按位与运算

6.8.2 位运算

■ 按位与 (&)

运算规则

$$0 \ \& \ 0 = 0$$

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

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

$$1 \ \& \ 1 = 1$$

用途

- 用 $0 \ \& \ x = 0$, 将一个数值中的某些位清0
- 用 $1 \ \& \ x = x$, 检验一个数值中某些位的值

例, $4 \ \& \ 6 = 4$

```
00000000 00000000 00000000 00000100
& 00000000 00000000 00000000 00000110
-----
00000000 00000000 00000000 00000100
```

按位与 (&)：利用按位与运算将某些位清零

设计一个数。对应于要清的位的值是0，其他位的值是1

例：将短整型数
32767的高字节中
的位数全部清0，保
留低字节部分的值。

解：将32767与短整型
数0xff进行位与运算

```
01111111 11111111
& 00000000 11111111
-----
00000000 11111111
```


按位与 (&): 用按位与运算检验某些位的值

设计一个数。对应于所要检验的位的值是1，其他位的值是0。如果结果为0，则检验位的值为0，否则为1。

例：检验短整型数
32767的最高位值
是0。

解：将32767与
0x8000执行按位与。

```
      01111111  11111111
&    10000000  00000000
-----
      00000000  00000000
```

6.8.2 位运算

■ 按位或 (|)

运算规则

$$0 \mid 0 = 0$$

$$1 \mid 0 = 1$$

$$0 \mid 1 = 1$$

$$1 \mid 1 = 1$$

用途

- 利用按位或运算将某些位置1

例, $4 \mid 6 = 6$

```
00000000 00000000 00000000 00000100
& 00000000 00000000 00000000 00000110
-----
00000000 00000000 00000000 00000110
```

按位或 (|): 利用按位或运算将某些位置位

- 设计一个数。对应于所要置位的位的值是1，其他位的值是0。
- 例：将一个字符型数据的最后4位设为1，其他位保持原状

解：将这个数与0x0f进行按位或，如这个数是10101010，则结果为：

10101010
00001111
10101111

6.8.2 位运算

■ 按位异或 (^)

运算规则

$$0 \wedge 0 = 0$$

$$1 \wedge 0 = 1$$

$$0 \wedge 1 = 1$$

$$1 \wedge 1 = 0$$

用途

- 利用按位异或将某些位取反
- 利用按位异或交换两个变量值

例, $4 \wedge 6 = 2$

```
00000000 00000000 00000000 00000100
& 00000000 00000000 00000000 00000110
-----
00000000 00000000 00000000 00000010
```

按位异或 (^): 利用按位异或将某些位取反

例, 需要将二进制位串01010101变成10100101, 即前四位取反。

解: 将这个位串
与11110000进
行按位异或。

```
      01010101
^     11110000
-----
      10100101
```

按位异或 (^): 利用按位异或交换两个变量值

交换a、b的值只要执行右侧语句:



```
a = a ^ b;  
b = b ^ a;  
a = a ^ b;
```

➔ 第二个语句等效于

$$b = b ^ (a ^ b) = b ^ b ^ a = 0 ^ a = a$$

➔ 第三个语句等效于

$$\begin{aligned} a &= (a ^ b) ^ (b ^ (a ^ b)) = (a ^ b) ^ a \\ &= a ^ a ^ b = 0 ^ b = b \end{aligned}$$

6.8.2 位运算

■ 取反 (~)

运算规则

$$\sim 0 = 1$$

$$\sim 1 = 0$$

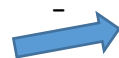
用途

- 将某数的每一位取反

■ ~和-运算的区别

- ~操作是把每一位都取反，即0变1，1变0
- -运算是把正数变成负数，负数变成正数
- 例：短整型变量x的值为32767，则-x的值为-32767，而~x的值为-32768

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1



1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1



1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

取反 (~): 将某数的每一位取反

- 将每一位取反可以用异或，为什么需要取反操作？

例：将0x85每一位取反可执行 $0x85 \wedge 0xff$

```
      01010101
^     11111111
-----
      10101010
```

- 取反运算的可移植性更好

例：将整型数a的每一位取反

- 如果整型数占2个字节，可以用 $a \wedge 0xffff$
- 但如果整型数占4个字节， $a \wedge 0xffff$ 只将后两个字节中的每一位取反，前两个字节保持原状。正确的做法是： $a \wedge 0xffffffff$
- 一个更好的解决方案： $\sim a$ ，在任何机器上都能得到正确的结果

取反 (~): 将某数的每一位取反

需求: 将某个数a除最后一位外的所有位取反

思路: 设计一个数, 最低位为0, 其他位为1。

方案一:

如果整数是16位表示,

可用 $a \wedge 0\text{xfffe}$ 。

如果整数是32位表示,

可用 $a \wedge 0\text{fffffff}$ 。

缺点: 影响程序的移植性。

方案二:

$a \wedge \sim 1$ 。

不管什么系统都能得到正确的结果。

6.8.2 位运算

■ 左移 (<<)

将一个数的二进制表示中的每个位向左移动若干位。

例如： $a = a \ll 2$ ；表示将a的二进制表示中的每一位向左移动两位，右边补0，左移后溢出的高位被舍弃。

- 若a的值为15，即二进制的00001111，执行这个操作后，a的值为60，即二进制的00111100。
- 若a的值为-127，即二进制的10000001，执行这个操作后，a的值为4，即二进制的00000100。

左移 (<<) 的应用

- ▶ 向左移动1位表示乘2。
- ▶ 由于移位操作比乘法操作执行速度快，所以有经验的程序员经常会用左移操作代替乘2的操作。
- ▶ 15向左移动2位表示 $15 * 2 * 2 = 60$ ，-1左移2位变成了-4。

6.8.2 位运算

■ 右移 (>>)

将一个数的二进制表示中的每个位向右移动若干位。

例如： $a = a >> 2$ ；表示将a的二进制表示中的每一位向右移动两位。

左边的填充：

- 无符号整数和正整数，左边补0。
- 负数，取决于编译环境。若系统采用逻辑右移，则填充0。若系统采用的是算术右移，则填充1。

6.8.2 位运算

■ 右移 (>>)

在绝大多数编译环境下，对**无符号数**采用**逻辑右移**，对**有符号数**采用**算术右移**。

例如：

- 无符号整数和正整数，左边补0。
- 负数，取决于编译环境。若系统采用**逻辑右移**，则填充0。若系统采用的是**算术右移**，则填充1。

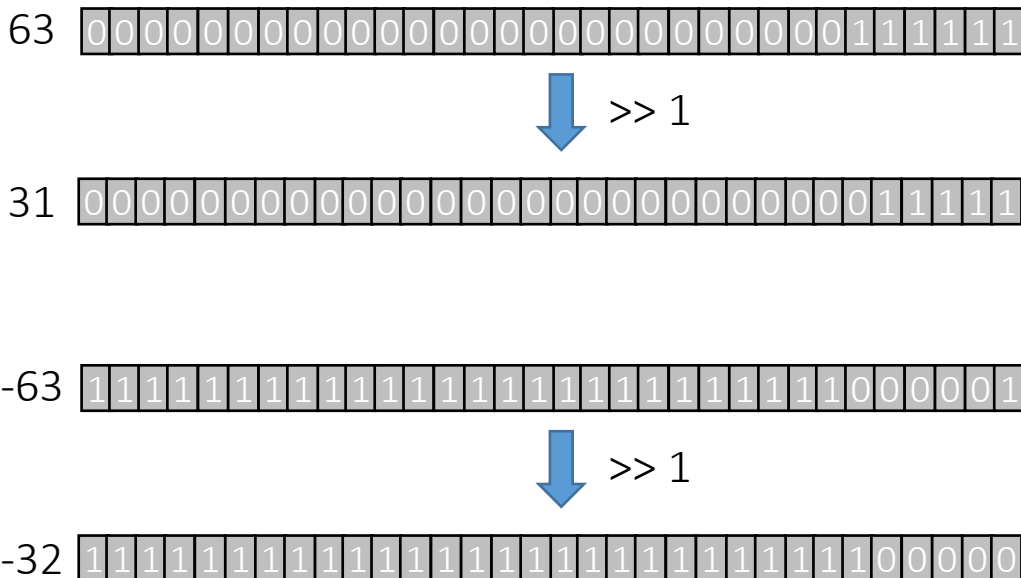
右移 (>>) 的应用



- ▶ 向右移动1位表示除2
- ▶ 由于移位操作比除法操作执行速度快，所以有经验的程序员经常会用左移操作代替除2的操作。

```
1. int main()
2. {
3.     int a = 63;
4.     int b = 63 >> 1;
5.     int c = -63;
6.     int d = (-63) >> 1;
7.     printf("%d %d %x %x\n", a, b, a, b);
8.     printf("%d %d %x %x\n", c, d, c, d);
9.     return 0;
10.}
```

```
63 31 3f 1f
-63 -32 ffffffff1 ffffffff0
```



练习

- 1、去掉最后一位 101101 -> 10110
- 2、在最后加一个0 101101 -> 1011010
- 3、在最后加一个1 101101 -> 1011011
- 4、把最后一位变成0 101101 -> 101100
- 5、把最后一位变成1 101100 -> 101101
- 6、把最后一位取反 101101 -> 101100
- 7、把右数第k位变成1 k=3 101001 -> 101101
- 8、把右数第k位变成0 k=3 101101 -> 101001
- 9、右数第k位去反 k=3 101001 -> 101101
- 10、取末三位 1101101 -> 101
- 11、取末k位 k=5 1101101 -> 01101
- 12、取右数第k位 k=4 1101101 -> 1
- 13、把右边连续的1变成0 100101111 -> 100100000
- 14、把右边连续的0变成1 11011000 -> 11011111

综合应用：灯的控制

【问题】假如房间里有8盏灯，用计算机控制。每个灯用一个二进制位表示。0表示关灯，1表示开灯。程序可以设置8盏灯的初始状态，可以将某一盏灯的状态反一反，即将原来开着的灯关掉或将原来关着的灯打开。

【方案】**用一个字节表示8盏灯的状态**



位运算

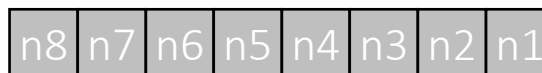


位域

综合应用：灯的控制

■ 用位运算实现

```
1. int main()
2. {
3.     int n1, n2, n3, n4, n5, n6, n7, n8, num, newStatus;
4.     char flag = 0;          /* 灯的状态 */
5.     printf("Please input the initial status of the lights (0 for off, 1 for on):");
6.     scanf("%d%d%d%d%d%d%d", &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8);
7.     flag = n1;
8.     flag |= (n2 << 1);
9.     flag |= (n3 << 2);
10.    flag |= (n4 << 3);
11.    flag |= (n5 << 4);
12.    flag |= (n6 << 5);
13.    flag |= (n7 << 6);
14.    flag |= (n8 << 7);
15.    printf("Please input the light number whose status is to be changed: ");
16.    scanf("%d", &num);
17.    --num;
18.    flag ^= (1 << num );
19.    newStatus = flag & (1 << num);
20.    newStatus = newStatus >> num;
21.    printf("新的状态是: %d\n", newStatus);
22.    return 0;
23.}
```



综合应用：灯的控制

■ 用位域实现

```
1. struct data {
2.     unsigned char n1: 1; unsigned char n2: 1; unsigned char n3: 1; unsigned char n4: 1;
3.     unsigned char n5: 1; unsigned char n6: 1; unsigned char n7: 1; unsigned char n8: 1;
4. };

5. int main()
6. {
7.     struct data flag;
8.     int n1, n2, n3, n4, n5, n6, n7, n8;
9.     printf("Please input the initial status of the lights (0 for off, 1 for on):");
10.    scanf("%d%d%d%d%d%d%d%d", &n1, &n2, &n3, &n4, &n5, &n6, &n7, &n8);
11.    flag.n1 = n1;    flag.n2 = n2;    flag.n3 = n3;    flag.n4 = n4;
12.    flag.n5 = n5;    flag.n6 = n6;    flag.n7 = n7;    flag.n8 = n8;
13.
14.    printf("%d %d %d %d %d %d %d %d\n",
15.           flag.n1, flag.n2, flag.n3, flag.n4, flag.n5, flag.n6, flag.n7, flag.n8);
16.
17.    return 0;
18.}
```

思考：读入灯的初始状态为何不使用如下代码？

```
scanf("%d%d%d%d%d%d%d%d",
      &flag.n1, &flag.n2, &flag.n3, &flag.n4,
      &flag.n5, &flag.n6, &flag.n7, &flag.n8);
```