

# 一维数组

---

## 一、一维数组的创建和初始化

---

### (1) 一维数组的创建

数组是一组相同类型元素的集合。数组创建方式：

```
Type_t arr_name [const_n]

// type_t    是指数组的元素类型
// arr_name  是指数组名
// const_n   是一个常量表达式，用来指定数组的大小
```

数组创建实例：

```
//代码1

int arr1[10];
//int表示数组元素类型，arr1表示数组类型，10表述数组内元素个数
```

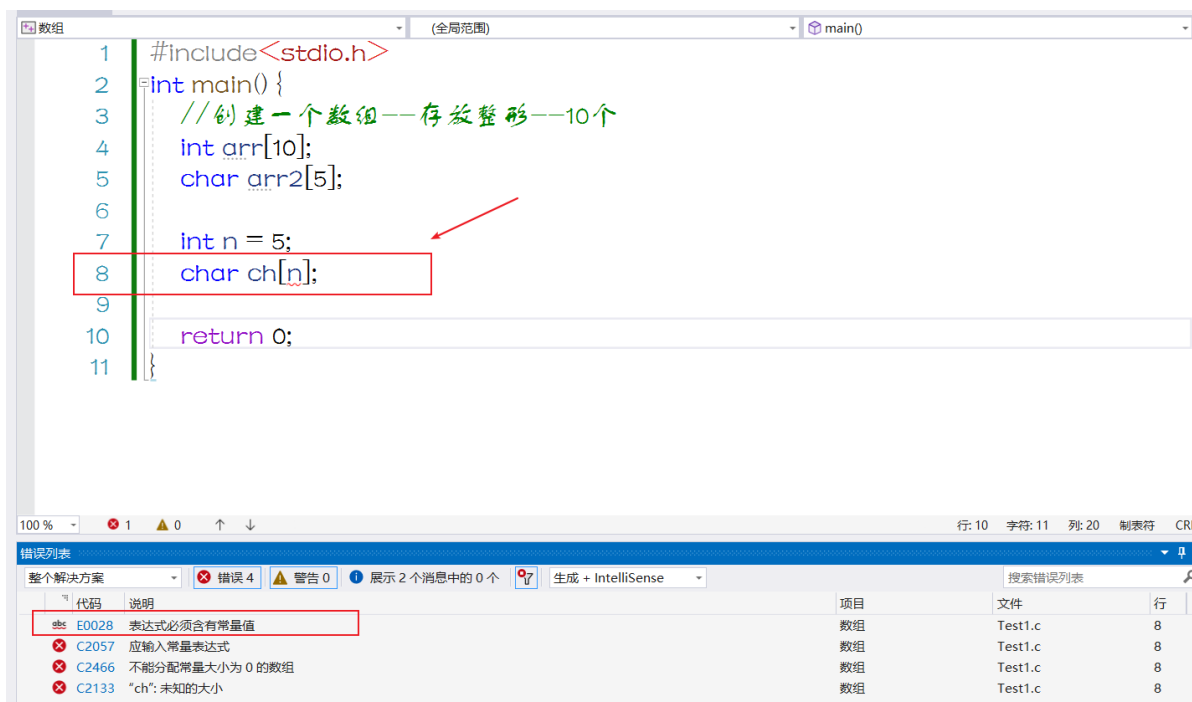
来看一个错误案例。

```
//代码2

int count=10;

int arr[count]; 数组可以正常创建? ---> 不可以
```

当我们运行之后，会出现这样的错误 应输入常量表达式。



//代码3

```
char arr3[10];
```

```
float arr4[1];
```

```
double arr5[20];
```

注:数组创建，【】里面要给一个常量才可以，不能使用变量。

## (2) 一维数组的初始化

在数组创建的同时给数组的内容一些合理的初始值（初始化）。

### 1) 整形数组初始化

```
int arr3[10]={1,2,3};
```

我们调试（按 F11）看一下，可以发现，数组里面前三个是存的数字，后面都是0。



```
char arr5[5]="abc";
```

The screenshot shows a C++ IDE with the following code:

```

11 //初始化
12 int arr3[10] = { 1,2,3 }; //不完全初始化, 剩下元素默认为0
13 char arr4[5] = { 'a','b' };
14 char arr5[5] = "ab";
15 return 0; 已用时间 <= 1ms
16 }
17

```

Below the code, the "自动窗口" (Auto Window) is open, displaying the "搜索" (Search) results for the variable `arr5`. The search depth is set to 3. The results show the memory address and value for `arr5` and its elements:

名称	值	类型
arr3	0x000000bc94aff4e8 {1, 2, 3, 0, 0, 0, 0, 0, 0, 0}	int[10]
arr4	0x000000bc94aff524 "ab"	char[5]
arr5	0x000000bc94aff544 "ab"	char[5]
[0]	97 'a'	char
[1]	98 'b'	char
[2]	0 '\0'	char
[3]	0 '\0'	char
[4]	0 '\0'	char

```
char arr6[5]={ 'a', 98};
```

[illegible]

#### ④不指定数组元素个数

```
char arr7[]="abcdef";
```

当我们并没有指定它的大小的时候，他会根据初始化的内容来确定数组有几个元素。

比如这里，有六个字符，加一个\0，共有7个元素。

```
16 char arr6[5]={ 'a', 98 };
17 char arr7[] = "abcdef";
18 return 0; 已用时间 <= 1ms
19 }
```

名称	值	类型
arr5	0x000000e7a414fa04 "ab"	char[5]
arr5[5]	-52 '?'	char
arr6	0x000000e7a414fa24 "ab"	char[5]
arr7	0x000000e7a414fa44 "abcdef"	char[7]
[0]	97 'a'	char
[1]	98 'b'	char
[2]	99 'c'	char
[3]	100 'd'	char
[4]	101 'e'	char
[5]	102 'f'	char
[6]	0 '\0'	char

### 3) sizeof与strlen

说一下常见的误区。

```
printf("%d\n", sizeof(arr7));
printf("%d\n", strlen(arr7));
```

先来输出看一下结果。

```
15 char arr5[5] = "ab";
16 char arr6[5]={ 'a', 98 };
17 char arr7[] = "abcdef";
18
19 printf("%d\n", sizeof(arr7));
20 printf("%d\n", strlen(arr7));
```

Microsoft Visual Studio 调试控制台

```
7
6
F:\C语言\C语言基础一代码\数组\x64\Debug\...
```

①**sizeof**是计算arr7所占空间的大小，刚才说了，arr7里面放了7个元素（6个字母，1个\n0）。

7个字符，每个字符大小是1 ---> 7\*1=7

②**strlen**是在求字符串长度，到\n0就停止了，并且\n0不算字符串的内容。

虽然放进去了abcdef\n0，但是求字符串长度的时候，遇到\n0就停止计算了，就是只算了6个字符。

**strlen**求的是\n0之前的字符个数。

## 👉 对比strlen 和 sizeof

1>**strlen** 和 **sizeof** 并没有什么关联

2>**strlen**是库函数，使用时要引用头文件；**sizeof**只是操作符。

3>**strlen**是求字符串长度的，只能针对字符串来求长度；**sizeof**是计算变量、数组、类型的大小，单位是字节。

## 4) 总结

当我们看到以下写法，都是正确的（上面演示过，这里总结一下）：

```
int arr1[10]={1,2,3};           // 不完全初始化
int arr2[]={1,2,3,4};           // 没有指定大小，是根据后边元素内容，确定数组大小
int arr3[5]={1,2,3,4,5};        // 完全初始化
char arr4[3]={'a',98,'c'};
char arr5[]={ 'a', 'b', 'c' };
char arr6[]="abcdef";
```

数组在创建的时候，如果想不指定数组的确定的大小就得初始化。

数组的元素个数根据初始化的内容来确定。

对于下面的代码要区分，内存中如何分配。

```
char arr8[] = "abc";
char arr9[] = { 'a', 'b', 'c' };
```

🔍分别用 **sizeof** 和 **strlen** 计算，结果分别是多少？

来分析一下。

arr8 数组里面的元素有：a、b、c、\0。

arr9 数组里面的元素有：a、b、c。

**sizeof**只关注所占空间的大小

arr8占了4个字节的空間。

arr9占了3个字节的空間。

**strlen**是求字符串长度的。（“\0”之前的字符个数）

arr8在“\0”之前有3个字符。（不包括“\0”）

arr9在“\0”之前有多少个字符是未知的，所以结果是个随机值。

来看一下运行结果：

```
char arr8[] = "abc";
char arr9[] = { 'a','b','c' };
printf("%d\n", sizeof(arr8));
printf("%d\n", sizeof(arr9));
printf("%d\n", strlen(arr8));
printf("%d\n", strlen(arr9));
```

4  
3  
3  
34

F:\C语言\C语言基  
要在调试停止时自  
按任意键关闭此窗

## 二、一维数组的使用

对于数组的使用，之前介绍了一个操作符：[]，下标引用操作符。它其实就是数组访问的操作符。

我们先写一个数组：

```
char arr[]="abcdef";
```

这个数组里面，有这样几个元素：[a][b][c][d][e][f][\0]。

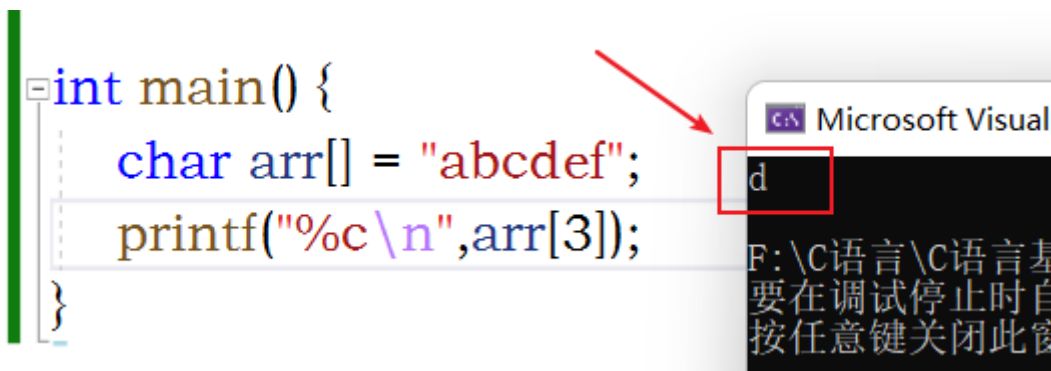
现在想拿出字符d。

就要用到下标。d是第4个元素，下标是3。（下标从0开始）

如果想要打印出它的内容，就要这样打印：

```
printf("%c\n",arr[3]);
```

看一下输出结果：



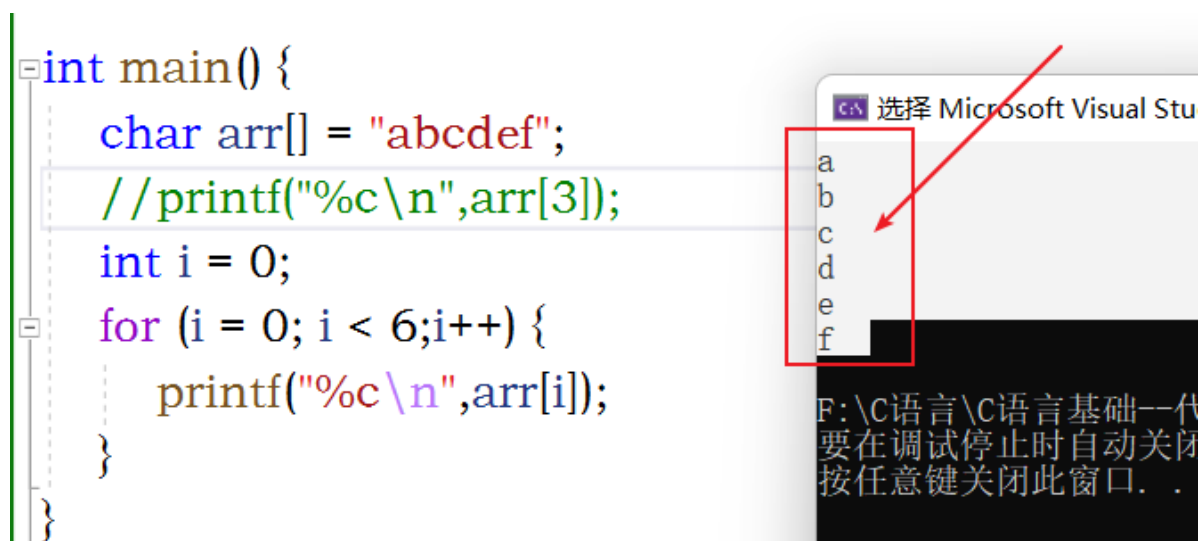
现在想将所有的元素输出。

该怎么办呢？

一个一个地输出是可以的，但是太麻烦。这里咱们用循环来做。

```
int i = 0;  
for (i = 0; i < 6; i++) {  
    printf("%c\n", arr[i]);  
}
```

看一下输出结果：



这里循环里边的6可以计算吗？

当然可以，可以用之前说过 `strlen` 来计算。

那么，这里的6，就可以用 `strlen(arr)` 来替代。

```
int i = 0;  
for (i = 0; i < strlen(arr); i++) {  
    printf("%c\n", arr[i]);  
}
```



同样，可以输出数组内元素。

```
int main() {  
    char arr[] = "abcdef";  
    //printf("%c\n",arr[3]);  
    int i = 0;  
    for (i = 0; i < strlen(arr); i++) {  
        printf("%c\n",arr[i]);  
    }  
}
```

选择 Microsoft

a  
b  
c  
d  
e  
f

F:\C语言\C语言  
要在调试停止时  
按任意键关闭此

⚠注意，这里可能会出一个警告。

有符号/无符号不匹配问题。

因为 `strlen` 函数返回的是一个**无符号整型**。

有警告的话可以强制类型转换一下。即把for循环第二个改为：`i < (int)strlen(arr)`

或者可以这样写：

```
int len = strlen(arr);  
for (i = 0; i < len; i++) {  
    printf("%c\n", arr[i]);  
}
```

看一下运行结果：

```
int main() {  
    char arr[] = "abcdef";  
    //printf("%c\n",arr[3]);  
    int i = 0;  
    int len = strlen(arr);  
    for (i = 0; i < len; i++) {  
        printf("%c\n", arr[i]);  
    }  
}
```

选择 Microsoft Visual St

a  
b  
c  
d  
e  
f

F:\C语言\C语言基础一  
要在调试停止时自动关  
按任意键关闭此窗口。

如果是个整型数组，也是可以算的。

我们不妨再来看个例子。

来写个数组：

```
int arr2[]={1,2,3,4,5,6,7,8,9,0};
```

要把每个元素打印出来。

先要知道元素个数：（用 sizeof 计算）

```
int sz=sizeof(arr2)/sizeof(arr2[0]);  
//用总大小除以每个元素的大小，就是元素的个数
```

整形数组，可以用sizeof来算。

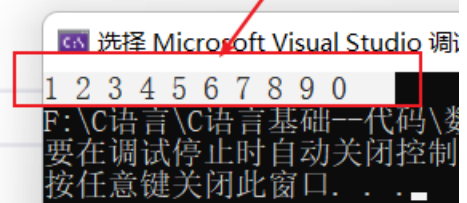
上面字符串数组，并不是不能用sizeof来算，而是用sizeof来算的话，包含了'\0'，不想要'\0'，就用strlen来算。

然后再用一个循环遍历一下数组，输出即可。

```
int i=0;  
for(i=0;i<sz;i++){  
    printf("%d ",arr2[i]);  
}
```

来看一下输出结果：

```
int arr2[] = { 1,2,3,4,5,6,7,8,9,0 };  
int sz = sizeof(arr2) / sizeof(arr2[0]);  
int i = 0;  
for (i=0; i < sz; i++) {  
    printf("%d ", arr2[i]);  
}
```



### 📦 总结

- 1.数组是使用下标来访问的，下标是从0开始。
- 2.数组的大小可以通过计算得到。

## 三、一维数组在内存中的存储

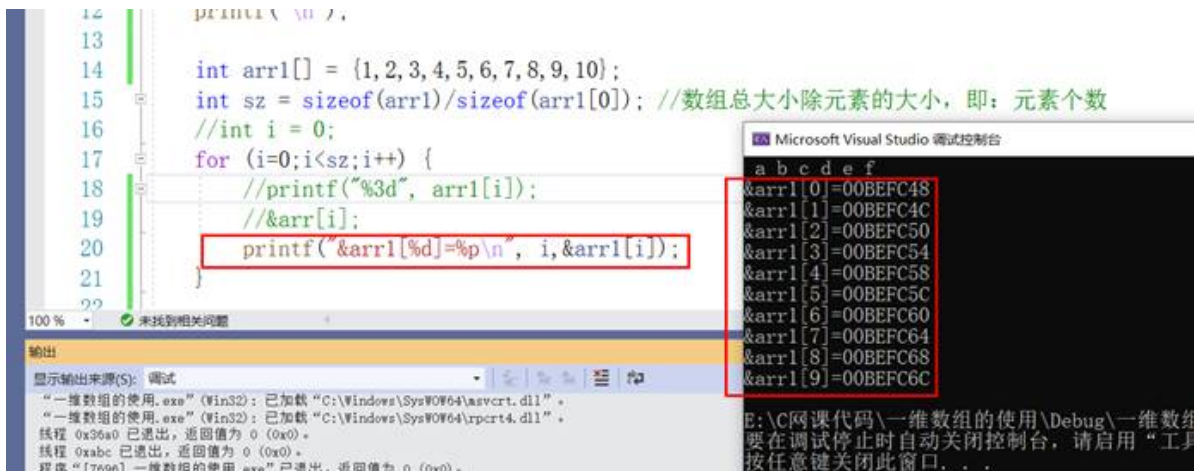
先来写一个数组：

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

现在想研究一下数组在内存中的布局，可以把每个元素的地址打印出来看一下。

```
int sz=sizeof(arr)/sizeof(arr[0]); //计算数组内有多少个元素
int i=0;
for(i=0;i<sz;i++){
    printf("&arr[%d]=%p\n",i,&arr[i]); //取出地址
}
```

看一下结果：



```
12  printf("%d\n", arr1[i]);
13
14  int arr1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
15  int sz = sizeof(arr1)/sizeof(arr1[0]); //数组总大小除元素的大小，即：元素个数
16  //int i = 0;
17  for (i=0;i<sz;i++) {
18      //printf("%3d", arr1[i]);
19      //&arr[i];
20      printf("&arr1[%d]=%p\n", i, &arr1[i]);
21  }
22
```

Microsoft Visual Studio 调试控制台

a	b	c	d	e	f
&arr1[0]	=	00BEFC48			
&arr1[1]	=	00BEFC4C			
&arr1[2]	=	00BEFC50			
&arr1[3]	=	00BEFC54			
&arr1[4]	=	00BEFC58			
&arr1[5]	=	00BEFC5C			
&arr1[6]	=	00BEFC60			
&arr1[7]	=	00BEFC64			
&arr1[8]	=	00BEFC68			
&arr1[9]	=	00BEFC6C			

E:\C网课代码\一维数组的使用\Debug\一维数组.exe: 要在调试停止时自动关闭控制台，请启用“工具”菜单中的“调试”子菜单中的“在调试停止时关闭控制台”选项。按任意键关闭此窗口。...

来看一下地址的规律，这里抽出最后两位，可以看到地址之间相隔4。

注意：下边有点小错误，是逢16进1，下边演示写错了，忘记改了。

```
&arr1[0]=00BEFC48
&arr1[1]=00BEFC4C
&arr1[2]=00BEFC50
&arr1[3]=00BEFC54
&arr1[4]=00BEFC58
&arr1[5]=00BEFC5C
&arr1[6]=00BEFC60
&arr1[7]=00BEFC64
&arr1[8]=00BEFC68
&arr1[9]=00BEFC6C
```

每一对相邻元素之间都差4（每个元素都是整型→4个字节）

一维数组在内存中连续存放，因为他们地址之间差4，而4又是一个字节的大小。

十进制：0~9  
十六进制：0~9、a、b、c、d、e、f

48	
4C (412)	> 差4
50 (逢10进1)	> 差4
54	
58	
5C (512)	
60 (逢10进1)	
64	
68	
6C (612)	

0 1 2 3 4 5 6 7 8 9

低地址 高地址

为什么会相差4？

因为每个元素都是整型，整型是4个字节。

数组在内存中是连续存放的。

随着下标的增长，元素的地址，也有规律地递增。一维数组在内存中连续存放。

## 二维数组

### 一、二维数组的创建和初始化

#### (1) 二维数组的创建

```
int arr[3][4];  
char arr[3][5];  
double arr[2][4];
```

二维数组表现形式（假想）：

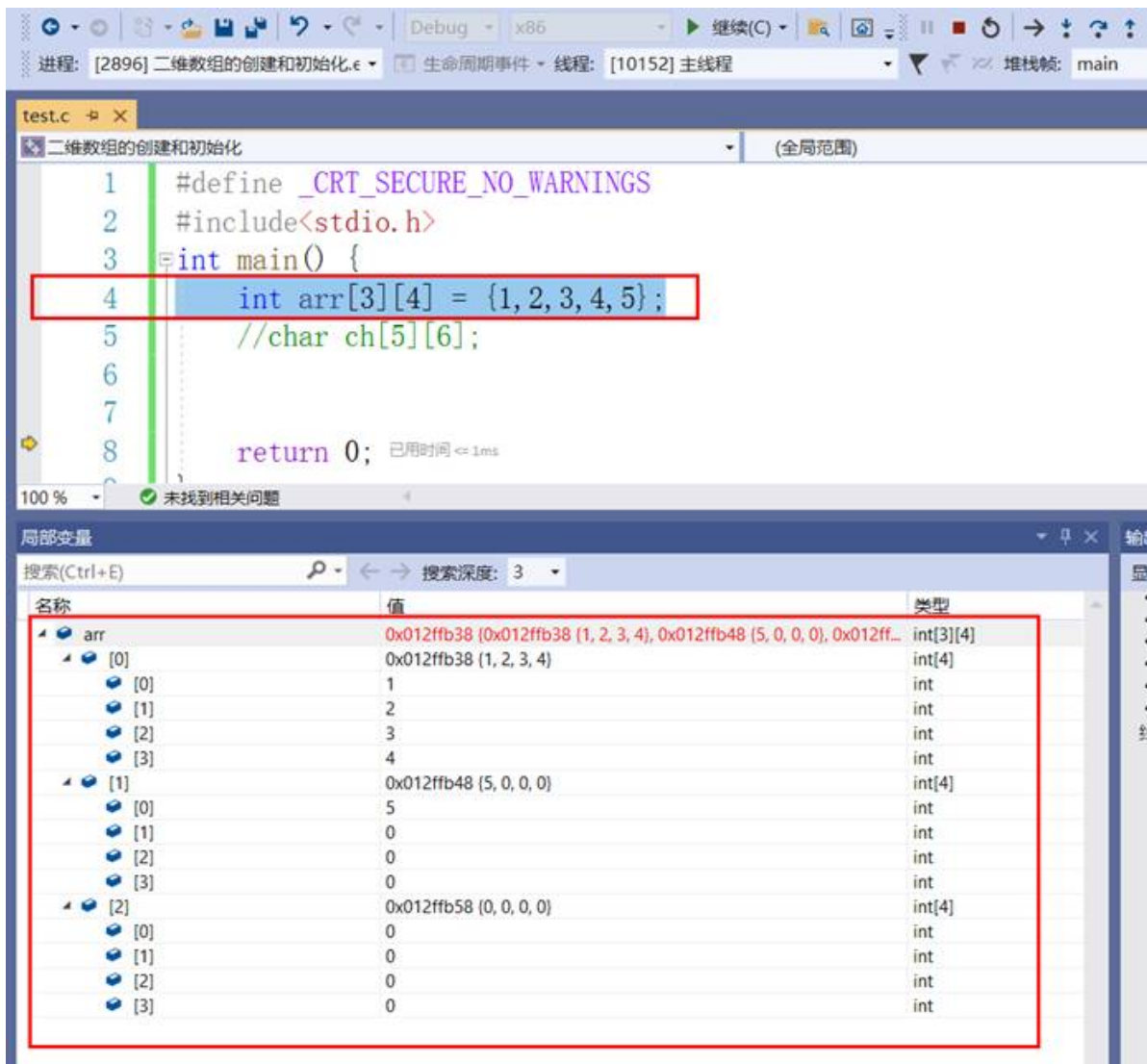


#### (2) 二维数组初始化

刚才的数组，咱们给它初始化一下。

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

看一下内存如何存储的：



★ 我们可以看到：

当二维数组进行 不完全初始化 的时候，后面的值也默认初始化为0。

一行放满之后，才会放第二行。

? 问：能否将123放在第一行，45放在第二行？

看内存图，我们可以发现：

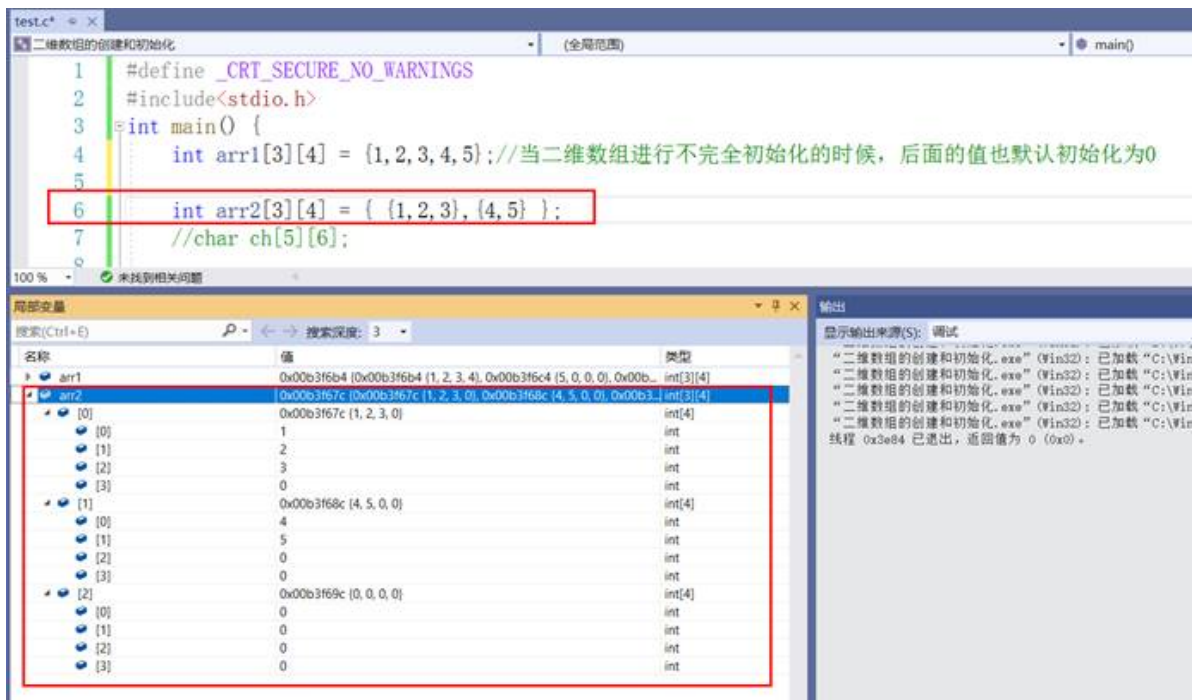
第一行可以当成一维数组；第二行也可以当成一维数组；第三行也可以当成一维数组。

我们可以把它当做一维数组来初始化！

即：

```
int arr2[3][4]={1,2,3},{4,5};
```

看一下内存如何存储的：

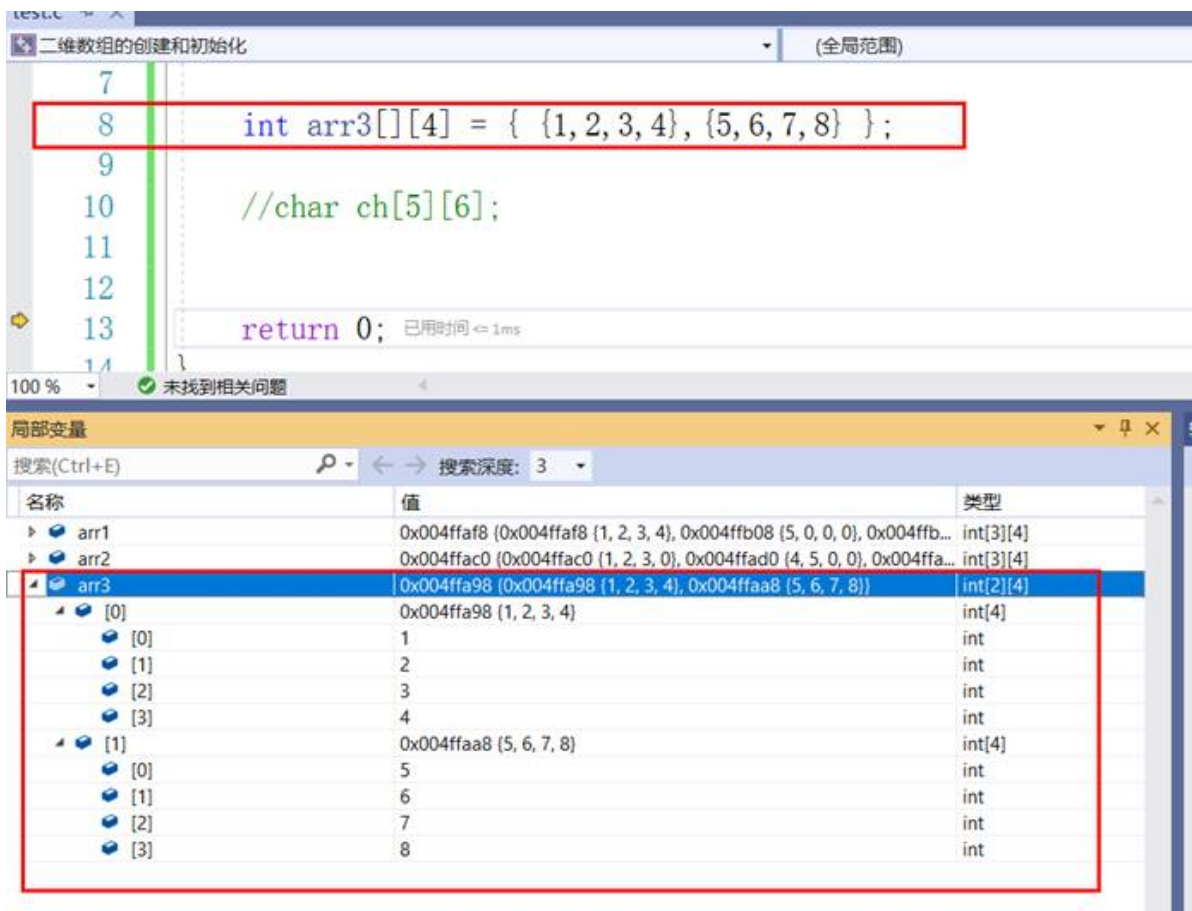


之前说过一维数组的初始化，可以不写元素个数，就像这样：

```
int arr[]={1,2,3,4}; //靠数组元素的个数，来确定数组大小
```

那二维数组呢？可以省略吗？

**二维数组 行可以省略，列不能省略！**（可以自己试一试，列省略的话编译出错，因为省略列就无法唯一确定数组存储方式）





## 二、二维数组的使用

二维数组的使用依然通过下标来实现。

举个例子。

先写个二维数组：

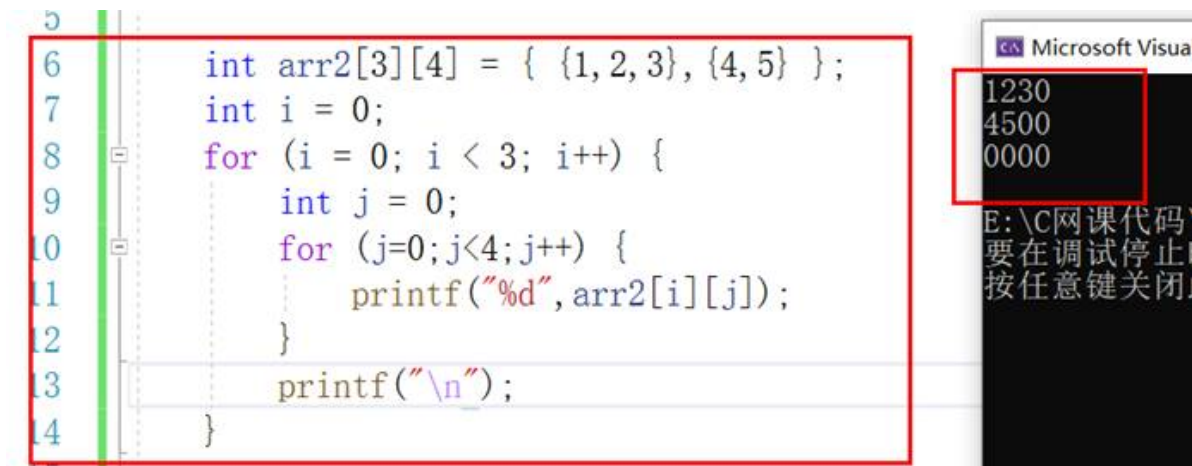
```
int arr2[3][4]={{1,2,3},{4,5}};
```

我们要将它元素输出。

因为要访问行和列，就要用到循环嵌套。

```
int i=0;
for(i=0;i<3;i++){    //遍历了行
    int j=0;
    for(j=0;j<4;j++){    //遍历了列
        printf("%d",arr2[i][j]);    //打印输出
    }
    printf("\n");    //换行
}
```

看一下输出结果：

The image shows a screenshot of a code editor with a red border. On the left, the C code is displayed with line numbers 6 through 14. The code defines a 2D array 'arr2' and uses nested loops to print its elements. On the right, a black console window titled 'Microsoft Visual' shows the output: '1230', '4500', and '0000' on three separate lines. A red box highlights the output text in the console window.

```
6 int arr2[3][4] = { {1, 2, 3}, {4, 5} };
7 int i = 0;
8 for (i = 0; i < 3; i++) {
9     int j = 0;
10    for (j=0;j<4;j++) {
11        printf("%d",arr2[i][j]);
12    }
13    printf("\n");
14 }
```

Microsoft Visual

1230  
4500  
0000

E:\C网课代码\  
要在调试停止时  
按任意键关闭

## 三、二维数组在内存中的存储

要知道，二维数组在内存中如何存储的。我们就可以将它们的地址全部打印出来看看。

打印输出地址：

```
printf("&arr[%d][%d]=%p",i,j,arr2[i][j]);    //打印输出地址
```

加上循环的完整代码：

```
int arr2[3][4]={{1,2,3},{4,5}};
int i=0;
for(i=0;i<3;i++){    //遍历了行
    int j=0;
    for(j=0;j<4;j++){    //遍历了列
        printf("&arr[%d][%d]=%p\n",i,j,&arr2[i][j]);    //打印输出地址
    }
}
```

可以发现,

**二维数组在内存中是一块连续的空间，可以看成由一维数组组成的数组**

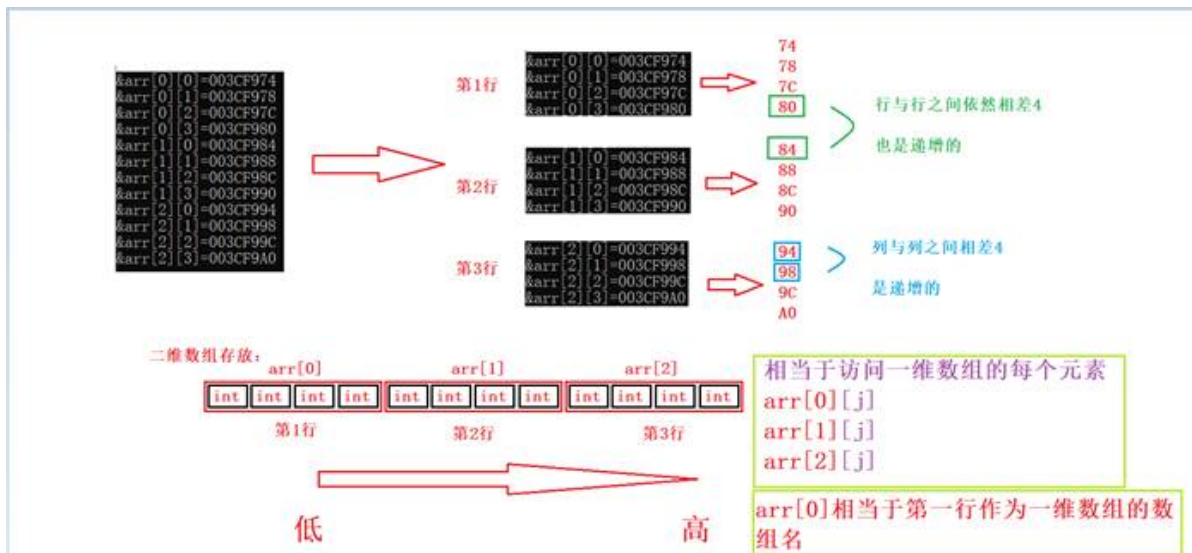
```
5
6 int arr2[3][4] = { {1, 2, 3}, {4, 5} };
7 int i = 0;
8 for (i = 0; i < 3; i++) {
9     int j = 0;
10    for (j=0;j<4;j++) {
11        //printf("%d", arr2[i][j]);
12        printf("&arr[%d][%d]=%p\n", i, j, &arr2[i][j]);
13    }
14    //printf("\n");
15 }
```

Microsoft Visual Studio 调试控制台  

```
&arr[0][0]=003CF974
&arr[0][1]=003CF978
&arr[0][2]=003CF97C
&arr[0][3]=003CF980
&arr[1][0]=003CF984
&arr[1][1]=003CF988
&arr[1][2]=003CF98C
&arr[1][3]=003CF990
&arr[2][0]=003CF994
&arr[2][1]=003CF998
&arr[2][2]=003CF99C
&arr[2][3]=003CF9A0
```

这里作图解释一下具体的内存结构。

之前说过假想的二维数组内存存储结构，这里其实存储的更加简单。



arr[0] 可以当作二维数组第一行的数组名。

arr[1] 可以当作二维数组第二行的数组名。

arr[2] 可以当作二维数组第三行的数组名。

**整个二维数组，就可以当作是三个一维数组组合而来的。**



# 数组作为函数参数

## 一、引子

往往我们在写代码的时候，会将数组作为参数传个函数，比如：

我要实现一个冒泡排序（这里要讲算法思想）函数将一个整型数组排序。那我们将会这样使用函数：

【写一个函数，函数的参数部分是一个数组。】

## 二、案例

### (1) 主要思想

案例：（冒泡排序）

先说一下冒泡排序的方法思想，这里要将下边的一行数升序排列。

拿第一项（10）与后边一项（9）比较，若后边的那一项小于前面的一项，就互换位置。然后再继续比较后一项（8），以此类推。

简单来说，就是两两相邻的元素进行比较。

冒泡排序



通过分析：

10个元素---> 9趟冒泡排序

第1趟 ---> 10个元素，要进行9次比较

第2趟 ---> 9个元素（最后一个已经不用动了），要进行8次比较

第3趟 ---> 8个元素，要进行7次比较

.....

## (2) 实操

### 1) 整体架构

了解完冒泡排序的基本思想之后，我们来做个小案例。

这里给出一个数组：

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

现在对这个数组进行排序，排成升序。

这里写一个新的函数 `bubble_sort()`，即冒泡排序函数。

既然是对数组 `arr` 进行排序，那么就需要将这个数组传递过去：`bubble_sort(arr)`，然后对这个函数进行设计。

还需要什么参数呢？不知道，先来设计一下函数吧。

### 2) 函数设计

这个函数的参数是什么？既然传过来是一个数组，那么可以拿一个数组（`int arr[]`）来接收，数组元素个数可以不指定。

函数排完序就行，不需要返回什么值，那么函数类型就是 `void`。如下：

```
void bubble_sort(int arr[]){  
  
}
```

然后写函数内容，上面对冒泡排序已经解释的很清楚了，接下来来写代码。

先确定冒泡排序趟数，趟数和元素有关系。

上面举例中，10个元素要跑9趟，那么  $n$  个元素就要跑  $(n-1)$  趟。

那么我们就需要知道元素个数，元素个数怎么算？讲一维数组的时候说过，这样算：

```
int sz=size(arr)/sizeof(arr[0]);
```

这样算可以吗？？不妨试一下。

传参的时候，只传递了一个数组上来，函数只拿了一个数组接收。

```
int i=0;  
int sz=sizeof(arr)/sizeof(arr[0]); //10个元素  
for(i=0;i<sz-1;i++){ //趟数 (9)  
  
}
```

这个 `for` 循环是趟数，接下来我们要在这个 `for` 循环里面，写每一趟的过程。

每一趟，就是拿相邻元素比较。

这里是升序排列，如果第一个元素比第二个元素大，就交换。

那么，循环条件是什么？上面分析的时候说过：

10个元素---> 9趟冒泡排序

第1趟 (i=0) ---> 10个元素，要进行9次 (sz-1) 比较

第2趟 (i=1) ---> 9个元素（最后一个已经不用动了），要进行8次 (sz-2) 比较

第3趟 (i=2) ---> 8个元素，要进行7次 (sz-3) 比较

.....

i ---> 要进行 (sz-1-i) 比较

由此可见，每一趟需要比较的次数是 `sz-1-i`。

```
int j=0;
for(j=0;j<sz-1-i;j++){ //每一趟做的事(相邻元素比较次数)
    if(arr[j]>arr[j+1]){ //比较相邻元素大小，若满足条件就交换
        int tmp=arr[j];
        arr[j]=arr[j+1];
        arr[j+1]=tmp;
    }
}
```

### 3) 遍历输出

函数部分就写完了，接下来在主函数中遍历输出一下排序之后的数组。

```
int i=0;
int sz=sizeof(arr)/sizeof(arr[0]);
for(i=0;i<sz;i++){
    printf("%d ",arr[i]);
}
```

### 4) 运行

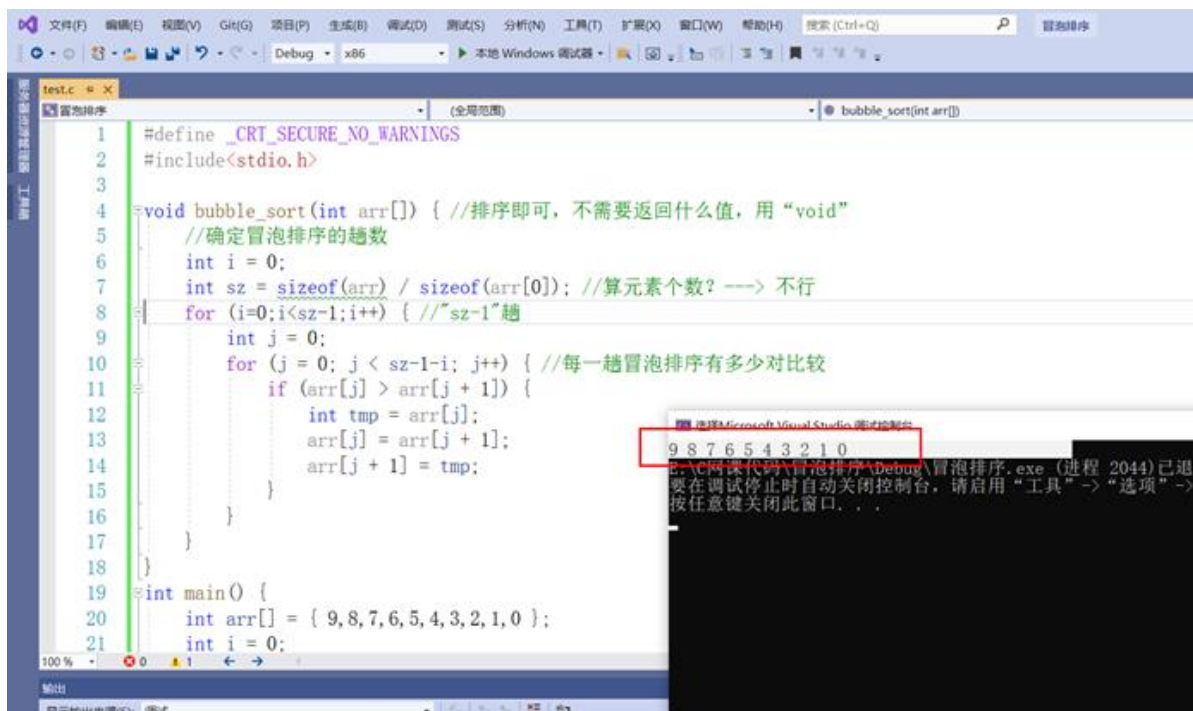
整体代码如下：

```
#include<stdio.h>
void bubble_sort(int arr[]){
    int i=0;
    int sz=sizeof(arr)/sizeof(arr[0]); //10个元素
    for(i=0;i<sz-1;i++){ //趟数(9)
        int j=0;
        for(j=0;j<sz-1-i;j++){ //每一趟做的事(相邻元素比较次数)
            if(arr[j]>arr[j+1]){ //比较相邻元素大小，若满足条件就交换
                int tmp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=tmp;
            }
        }
    }
}
```

```

    }
}
}
int main(){
    int arr[]={9,8,7,6,5,4,3,2,1,0};
    int i=0;
    int sz=sizeof(arr)/sizeof(arr[0]);
    for(i=0;i<sz;i++){
        printf("%d ",arr[i]);
    }
}
}

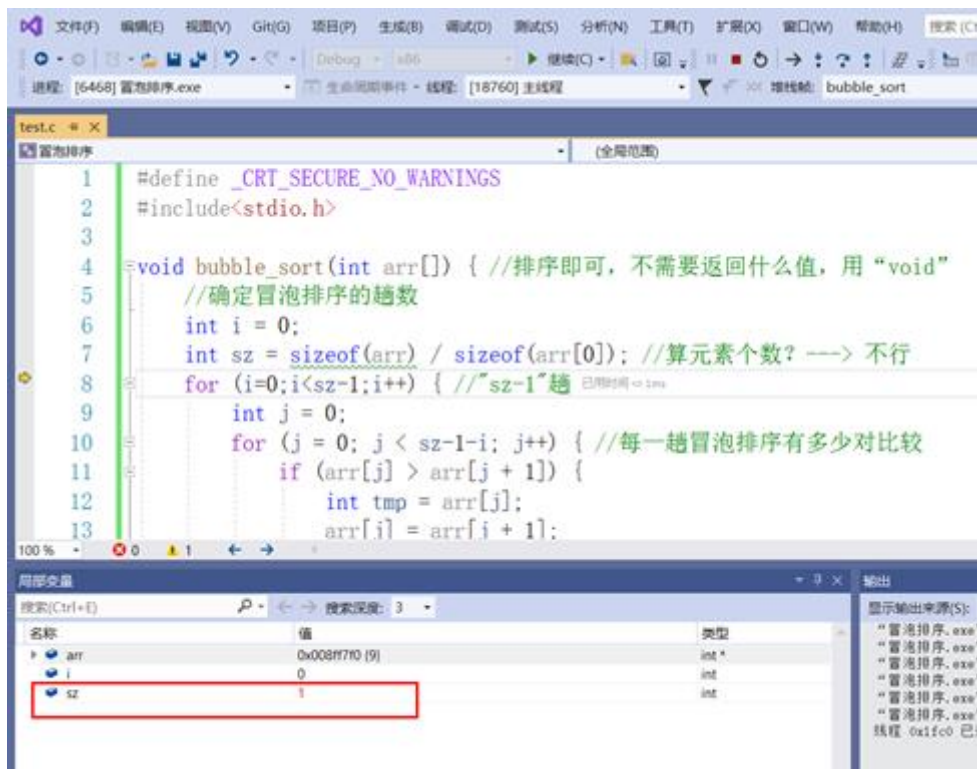
```



哎呀，并没有达到预想效果啊？

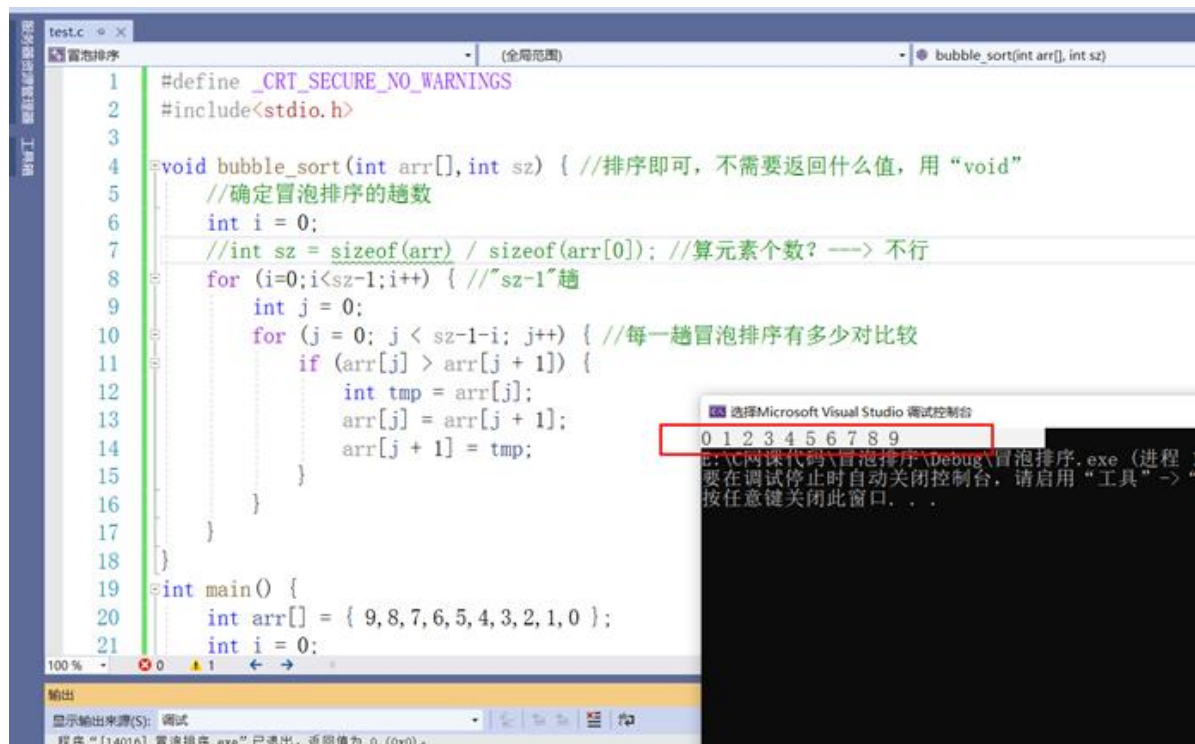
哪儿出错了？

我们来调试看一看：



arr是数组，我们对数组arr进行传参的时候。按照之前说的，实参传给形参的时候，形参是实参的一份临时拷贝。会产生空间浪费！传递数组的时候，传递的不是数组本身，而是数组首元素地址。

数组元素个数 sz 我们可以在外面算好，再传进去。



## 5)优化

如果刚开始整个数据就是顺序排列，我们就不需要再继续了。

这里可以增加一个flag变量来实现。

若要排的数组是这样的：912345678

第1趟： 192345678

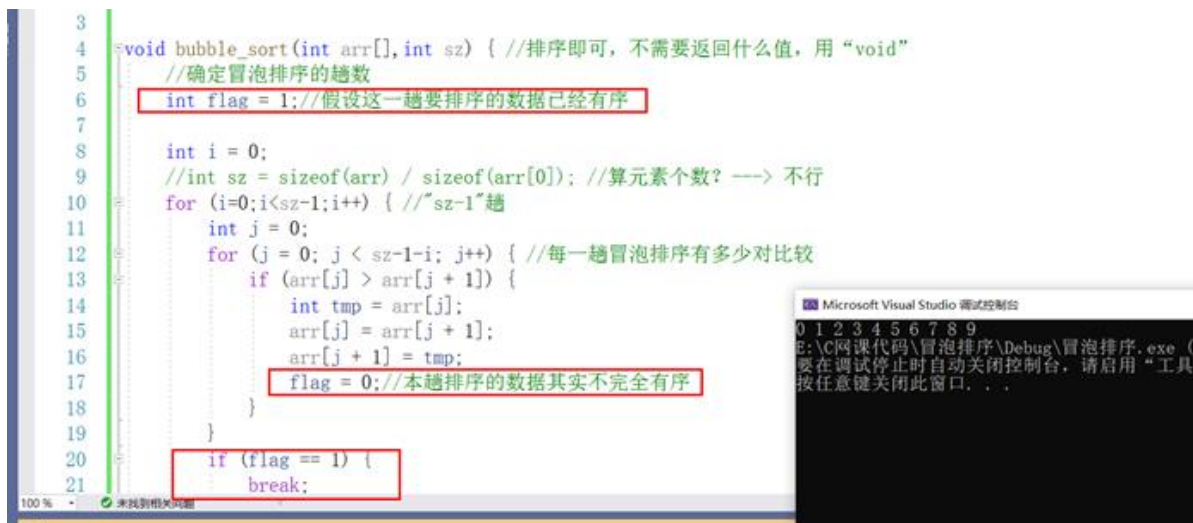
flag=0

123456789

第2趟： 123456789

flag=1

结束



The screenshot shows a C++ code editor with the following code and annotations:

```
3
4 void bubble_sort(int arr[], int sz) { //排序即可，不需要返回什么值，用“void”
5     //确定冒泡排序的趟数
6     int flag = 1; //假设这一趟要排序的数据已经有序
7
8     int i = 0;
9     //int sz = sizeof(arr) / sizeof(arr[0]); //算元素个数？ --> 不行
10    for (i = 0; i < sz - 1; i++) { //“sz-1”趟
11        int j = 0;
12        for (j = 0; j < sz - 1 - i; j++) { //每一趟冒泡排序有多少对比较
13            if (arr[j] > arr[j + 1]) {
14                int tmp = arr[j];
15                arr[j] = arr[j + 1];
16                arr[j + 1] = tmp;
17                flag = 0; //本趟排序的数据其实不完全有序
18            }
19        }
20        if (flag == 1) {
21            break;
22        }
23    }
24 }
```

Annotations (red boxes):

- Line 6: `int flag = 1; //假设这一趟要排序的数据已经有序`
- Line 17: `flag = 0; //本趟排序的数据其实不完全有序`
- Line 20: `if (flag == 1) {`

Output window (Microsoft Visual Studio 调试控制台):

```
0 1 2 3 4 5 6 7 8 9
E:\C网课代码\冒泡排序\Debug\冒泡排序.exe (
要在调试停止时自动关闭控制台，请启用“工具
按任意键关闭此窗口，...
```

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
void bubble_sort(int arr[], int sz) { //排序即可，不需要返回什么值，用“void”
```

```
    //确定冒泡排序的趟数
```

```

int flag = 1; //假设这一趟要排序的数据已经有序

int i = 0;

//int sz = sizeof(arr) / sizeof(arr[0]); //算元素个数? ---> 不行

for (i=0; i<sz-1; i++) { // "sz-1"趟

    int j = 0;

    for (j = 0; j < sz-1-i; j++) { //每一趟冒泡排序有多少对比较

        if (arr[j] > arr[j + 1]) {

            int tmp = arr[j];

            arr[j] = arr[j + 1];

            arr[j + 1] = tmp;

            flag = 0; //本趟排序的数据其实不完全有序

        }

    }

    if (flag == 1) {

        break;

    }

}

int main() {

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

    int i = 0;

    int sz = sizeof(arr) / sizeof(arr[0]);

    //对arr进行排序，排成升序

    bubble_sort(arr, sz); //冒泡排序函数，传递过去的是首元素地址: &arr[0]

    for (i = 0; i < sz; i++) {

```

```

    printf("%d ", arr[i]);

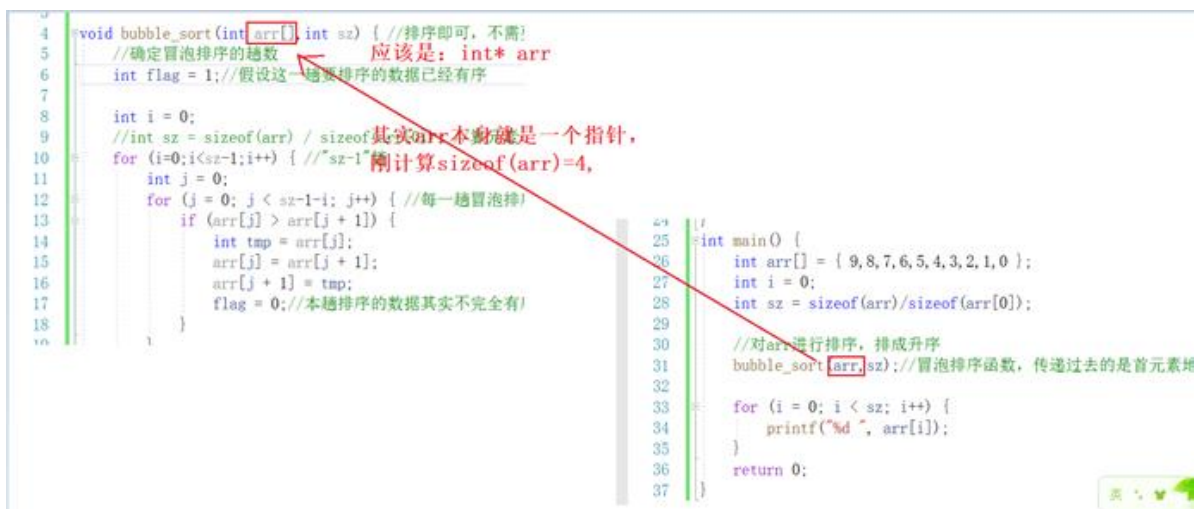
}

return 0;

}

```

传递数组的时候，传递的不是数组本身，而是数组首元素地址。当我们在函数内部需要元素个数的时候，必须在外边求好，以参数方式传进来。



```

1 void bubble_sort(int arr[], int sz) { //排序即可，不需
2     //确定冒泡排序的趟数
3     int flag = 1; //假设这一趟排序的数据已经有序
4
5     int i = 0;
6     //int sz = sizeof(arr) / sizeof(arr[0]);
7     for (i = 0; i < sz - 1; i++) { // "sz-1" 表示
8         int j = 0;
9         for (j = 0; j < sz - 1 - i; j++) { //每一趟冒泡排
10             if (arr[j] > arr[j + 1]) {
11                 int tmp = arr[j];
12                 arr[j] = arr[j + 1];
13                 arr[j + 1] = tmp;
14                 flag = 0; //本趟排序的数据其实不完全有
15             }
16         }
17     }
18 }
19
20 int main() {
21     int arr[] = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
22     int i = 0;
23     int sz = sizeof(arr) / sizeof(arr[0]);
24
25     //对arr进行排序，排成升序
26     bubble_sort(arr, sz); //冒泡排序函数，传递过去的是首元素地址
27
28     for (i = 0; i < sz; i++) {
29         printf("%d ", arr[i]);
30     }
31     return 0;
32 }

```

Red annotations in the image:

- Line 1: `int arr[]` is highlighted with a red box. A red arrow points to it with the text: "应该是: `int* arr`".
- Line 26: `arr` is highlighted with a red box. A red arrow points to it with the text: "其实arr本就是一个指针，" and "刚计算sizeof(arr)=4,".

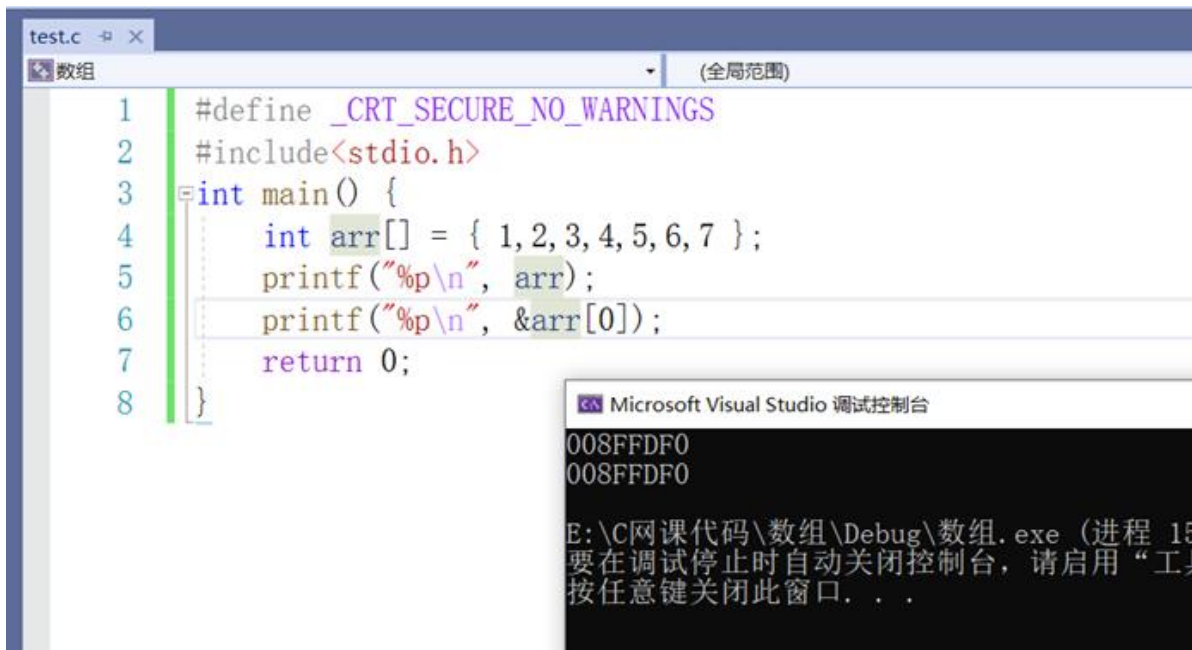
break语句只用于for和switch，在if语句中不能使用。因为if是循环语句，所以不能用break来结束。

但我们用的if语句是放在循环里面的，break是用在：当条件满足的时候，跳出外层for循环。

## 数组名是什么

数组名就是首元素地址





The screenshot shows a Visual Studio window with a C file named `test.c`. The code defines `_CRT_SECURE_NO_WARNINGS`, includes `<stdio.h>`, and has a `main` function that declares an array `arr` with values `{ 1, 2, 3, 4, 5, 6, 7 }`. It then prints the address of the array and the address of its first element, both using the `%p` format specifier. The debug console shows two identical memory addresses: `008FFDF0`. Below the addresses, a message indicates the program path and provides instructions on how to close the console window.

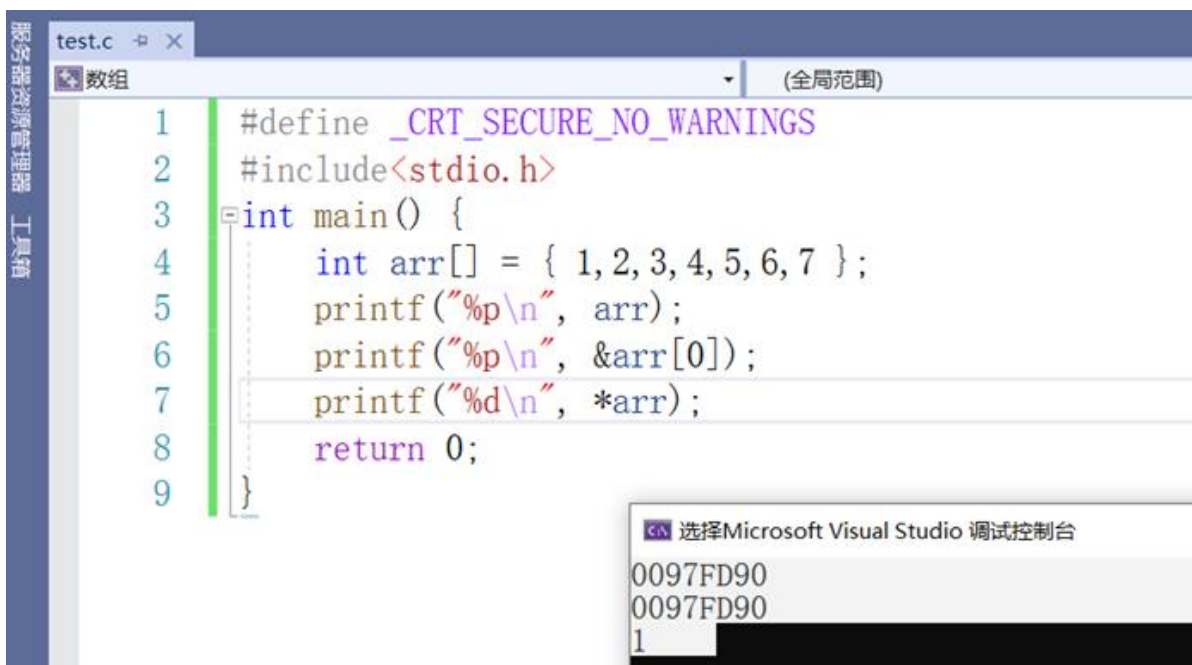
```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include<stdio.h>
3 int main() {
4     int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
5     printf("%p\n", arr);
6     printf("%p\n", &arr[0]);
7     return 0;
8 }
```

Microsoft Visual Studio 调试控制台

008FFDF0  
008FFDF0

E:\C网代码\数组\Debug\数组.exe (进程 15...)  
要在调试停止时自动关闭控制台，请启用“工具...  
按任意键关闭此窗口。...

再看看解引用之后的结果：



This screenshot shows the same Visual Studio window, but the code in `test.c` has been updated. A third `printf` statement, `printf("%d\n", *arr);`, has been added on line 7 to print the value of the first element of the array. The debug console now shows the same two memory addresses, `0097FD90`, followed by the value `1`, which is the first element of the array.

```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include<stdio.h>
3 int main() {
4     int arr[] = { 1, 2, 3, 4, 5, 6, 7 };
5     printf("%p\n", arr);
6     printf("%p\n", &arr[0]);
7     printf("%d\n", *arr);
8     return 0;
9 }
```

选择Microsoft Visual Studio 调试控制台

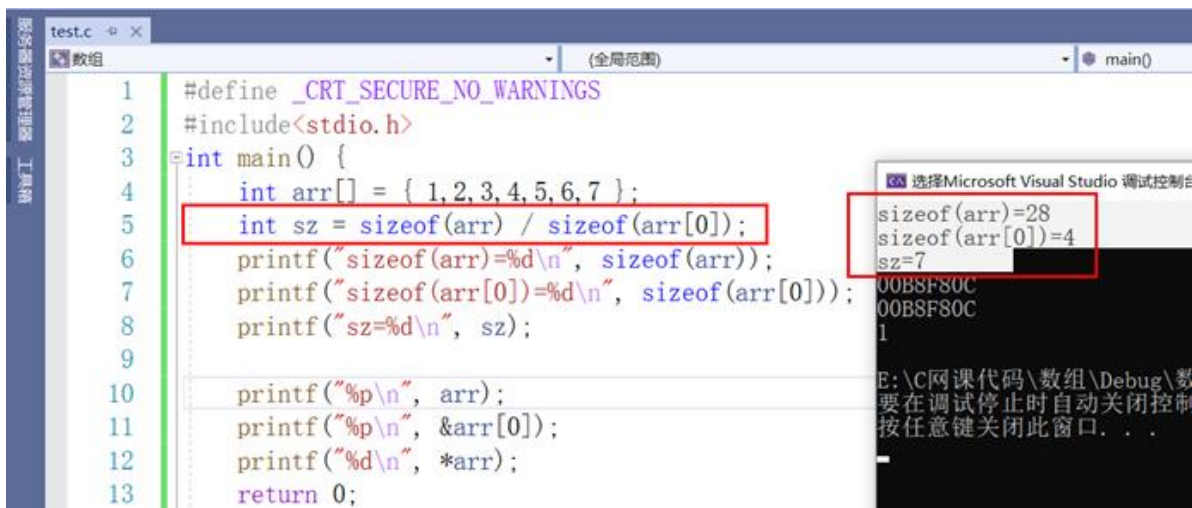
0097FD90  
0097FD90  
1

两个例外：（数组名代表的不是首元素地址）

#### 1、`sizeof`（数组名）

此时，数组名表示整个数组。

`sizeof`（数组名） 计算的是整个数组的大小，单位是字节。



## 2、&数组名

数组名代表整个数组。&数组名，取出的是整个数组的地址。



三个的结果一样，但是意义却不一样！

&arr 是整个数组的地址（所有元素都包含了），而 arr 和 &arr[0] 是数组的首元素地址。

问：数组首元素地址与整个数组地址有什么区别？

不妨来看一下下面的小案例：

```

10 printf("%p\n", &arr); //整个数组的地址
11 printf("%p\n", &arr + 1);
12
13 printf("%p\n", arr); //首元素地址
14 printf("%p\n", arr + 1);
15
16 printf("%p\n", &arr[0]); //首元素地址
17 printf("%p\n", &arr[0] + 1);
18
19 printf("%d\n", *arr);

```

```

sizeof(arr)=28
sizeof(arr[0])=4
sz=7
009EFA74
009EFA90
009EFA74
009EFA78
009EFA74
009EFA78
1
E:\C网代码\数组\Debug
要在调试停止时自动关闭
按任意键关闭此窗口. . .

```

74-->90, 怎么就加了16啊?(有待考究)

计算(补充): 1C--> 1-16, C-12, 16+12=28

7个元素, 每个元素4个字节, 应该加的是28啊? 为什么?

74-->90  
16

```

10 printf("%p\n", &arr); //整个数组的地址
11 printf("%p\n", &arr + 1);
12
13 printf("%p\n", arr); //首元素地址
14 printf("%p\n", arr + 1);
15
16 printf("%p\n", &arr[0]); //首元素地址
17 printf("%p\n", &arr[0] + 1);

```

&arr+1是跳过一个数组的地址

arr+1为下一个元素的地址

&arr[0]+1为下一个元素的地址

```

009EFA74
009EFA90
009EFA74
009EFA78
009EFA74
009EFA78

```

用DEV C++编译器却是对的:

```

数组.cpp
1 #include<stdio.h>
2 int main() {
3     int arr[] = { 1,2,3,4,5,6,7 };
4     int sz = sizeof(arr) / sizeof(arr[0]);
5     printf("sizeof(arr)=%d\n", sizeof(arr));
6     printf("sizeof(arr[0])=%d\n", sizeof(arr[0]));
7     printf("sz=%d\n", sz);
8
9     printf("%p\n", &arr); //整个数组的地址
10    printf("%p\n", &arr + 1);
11
12    printf("%p\n", arr); //首元素地址
13    printf("%p\n", arr + 1);
14
15    printf("%p\n", &arr[0]); //首元素地址
16    printf("%p\n", &arr[0] + 1);
17
18    printf("%d\n", *arr);
19    return 0;
20 }

```

```

C:\Users\lenovo\Desktop\数组.exe
sizeof(arr)=28
sizeof(arr[0])=4
sz=7
0000000000062FE00
0000000000062FE1C
0000000000062FE00
0000000000062FE04
0000000000062FE00
0000000000062FE04
1
Process exited after 0.3494 s
请按任意键继续. . .

```

代码:

```

#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>

```

```

int main() {

    int arr[] = { 1,2,3,4,5,6,7 };

    int sz = sizeof(arr) / sizeof(arr[0]);

    printf("sizeof(arr)=%d\n", sizeof(arr));

    printf("sizeof(arr[0])=%d\n", sizeof(arr[0]));

    printf("sz=%d\n", sz);


    printf("%p\n", &arr); //整个数组的地址

    printf("%p\n", &arr + 1);


    printf("%p\n", arr); //首元素地址

    printf("%p\n", arr + 1);


    printf("%p\n", &arr[0]); //首元素地址

    printf("%p\n", &arr[0] + 1);


    printf("%d\n", *arr);

    return 0;

}

```

总结:

1.sizeof(数组名), 计算整个数组的大小, sizeof内部单独放一个数组名, 数组名表示整个数组。

2.&数组名, 取出的是数组的地址。&数组名, 数组名表示整个数组。

除此两种情况之外, 所有的数组名都表示数组首元素的地址。

