

# 一、回顾

指针的主题，我们在[\(2条消息\) C语言基础-初识指针 雨翼轻尘的博客-CSDN博客](#)已经接触过了。我们知道了指针的概念：

- 1、 指针就是个变量，用来存放地址，地址唯一标识一块内存空间。
- 2、 指针的大小是固定的4/8个字节（32平台/64平台）。
- 3、 指针有类型，指针的类型决定了指针的+整数的步长，指针解引用操作的时候的权限。
- 4、 指针的运算。

这一章节，我们继续探讨。

首先再来说明一下指针大小的问题。

看如下代码，输出结果是多少呢？

```
#include<stdio.h>
void test(int arr[]) {
    int sz = sizeof(arr) / sizeof(arr[0]);
    printf("%d\n", sz);
}
int main() {
    int arr[10] = { 0 };
    test(arr);
}
```

分析一下这个函数：

```
void test(int arr[]) { //arr是指针变量
    int sz = sizeof(arr) / sizeof(arr[0]);
    //sizeof(arr)求指针大小-->4个字节(32平台)
    //sizeof(arr[0])是求一个元素的大小，整型-->4个字节
    //于是：sizeof(arr)/sizeof(arr[0])=4/4=1
    printf("%d\n", sz);
}
```

经过分析，输出结果是1。

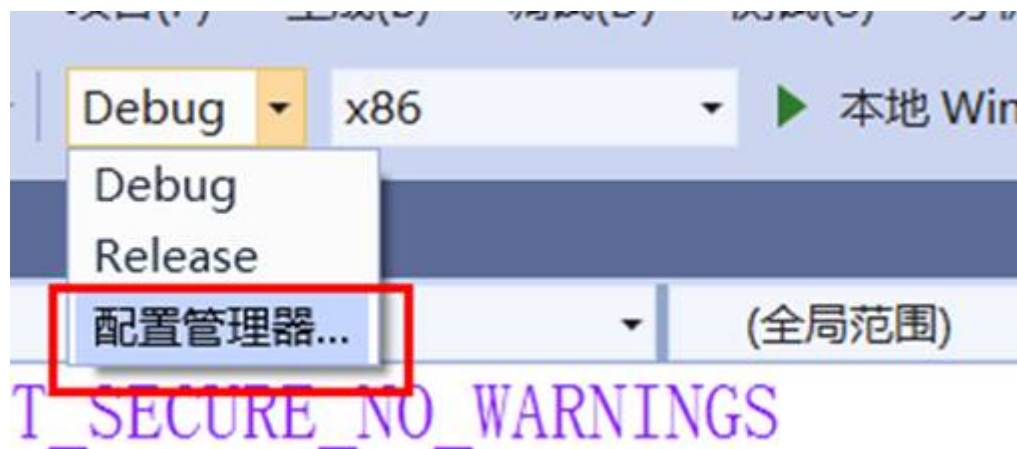
在编辑器里面运行也是1：



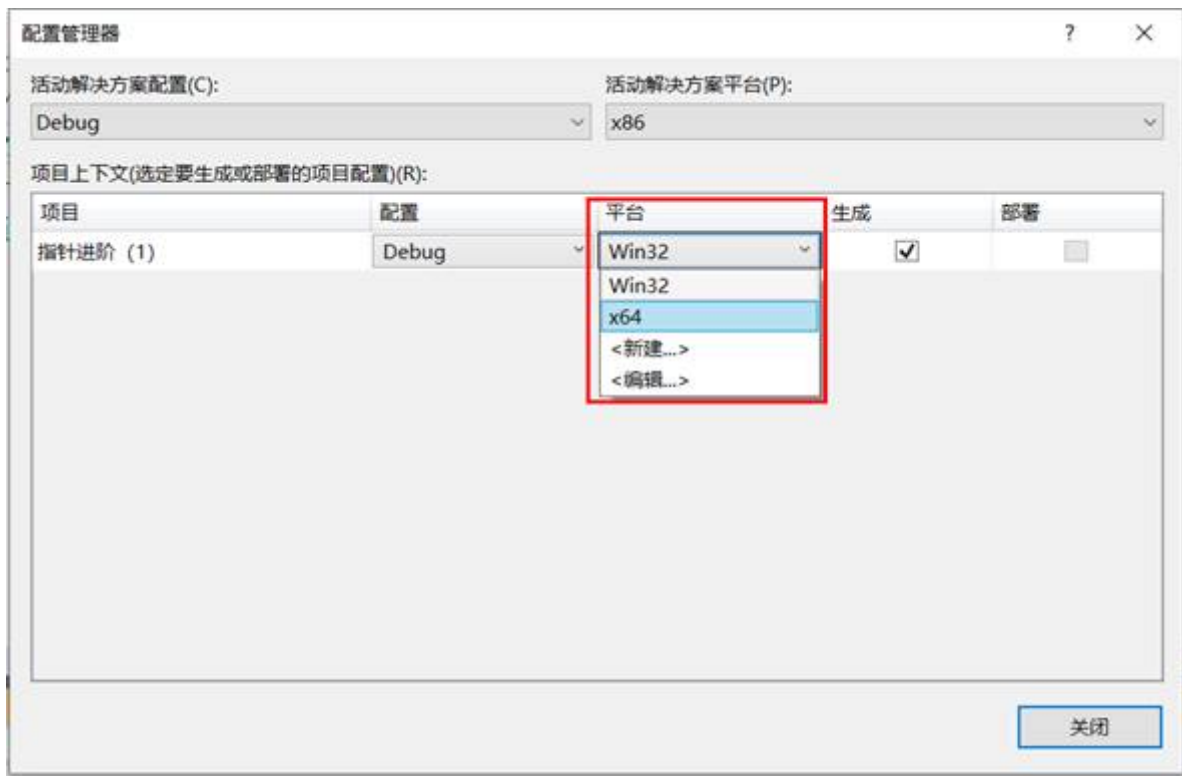
有的小伙伴说，我输出的是2啊？

别急，我配置一下这个地方。

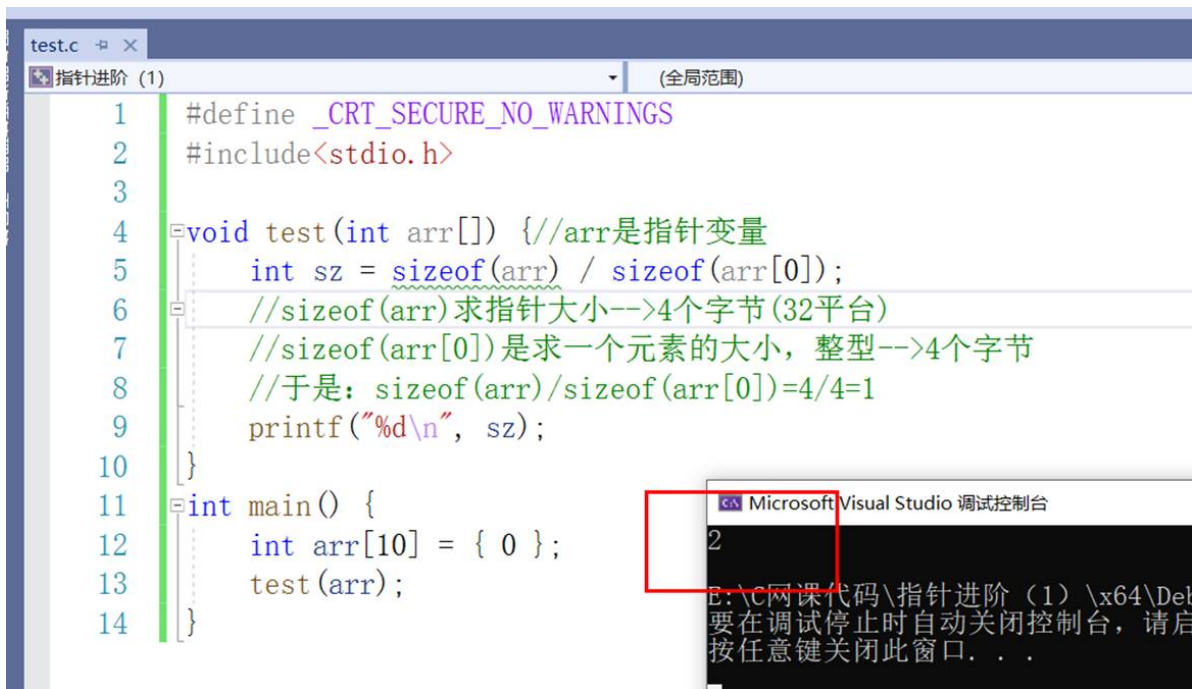
如图，打开配置管理器：



将平台改为x64平台：



这时候，再次运行，发现结果就是2：



简单分析一下：

```

void test(int arr[]) { //arr是指针变量
    int sz = sizeof(arr) / sizeof(arr[0]);
    //sizeof(arr)求指针大小-->8个字节(64平台)
    //sizeof(arr[0])是求一个元素的大小，整型-->4个字节
    //于是: sizeof(arr)/sizeof(arr[0])=8/4=2
    printf("%d\n", sz);
}

```

## 二、字符指针

在指针类型中我们知道有一种指针类型为**字符指针** `char*`。

### 1.基本用法

之前我们初识指针的时候，说过用法。

如下定义：

```
char ch = 'w'; //字符变量ch
char* pc=&ch;  //将字符变量ch的地址取出来，存在pc中。pc就被称为字符指针，类型就是char*
```

字符指针是一个指针变量，里面存放一个字符的地址。

将字符指针解引用，可以找到字符。

```
*pc = 'w';
```

### 2.误区

#### (1) 字符指针存放字符串首元素地址

看如下代码：

```
int main(){
    char* pstr="hello";
    printf("%s\n",pstr);
    return 0;
}
```

看第一行代码。

 这里，是把一个字符串"hello"存放到了pstr指针变量里面了吗？

 注意

代码 `char* pstr="hello";`，

特别容易让我们以为是把字符串hello放到了字符指针pstr里面。

但本质是把字符串hello的 **首字母地址** 存放到了pstr中。

其实一般这里有两种理解：

```

34 int main() {
35     char* p = "abcdef";
36
37     return 0;
38 }

```

理解一：

把“abcdef”放入p中。

“abcdef”共7个字节（包括\0），而p是指针变量，32平台下只有4个字节的存储空间。把“abcdef”放入p变量中，是存不下的。

这种理解错误！

理解二：

内存中放一个字符串“abcdef”，是一个常量字符串。

把a的地址赋给了p

第一种理解是错误的！

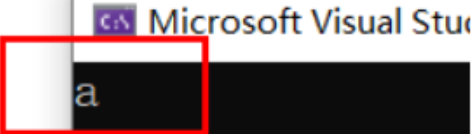
第二种理解是正确的。**a的地址赋值给了p**。（常量字符串有什么需要注意的地方，后边讲解。）

可以输出看一下，p指针解引用之后的结果：

```

34 int main() {
35     char* p = "abcdef";
36     printf("%c\n", *p);
37     return 0;
38 }

```



输出结果是a，p里面存放的是a的地址！

再想一个问题：

既然p里面存放的是a的地址，那么如果打印的话，是否能打印出来abcdef呢？

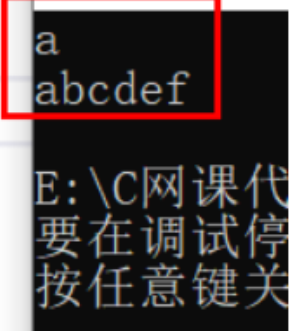
不妨试一下：

```
printf("%s\n", p);
```

```

34 int main() {
35     char* p = "abcdef";
36     printf("%c\n", *p);
37     printf("%s\n", p);
38     return 0;
39 }

```



将字符串赋值给一个字符指针变量p，不是把字符串的内容赋值给p，而是把字符串首字符的地址赋给了p。

后边讲[内存布局](#)的时候，给大家补充一下为何输出结果是这样的。

## (2) 输出问题

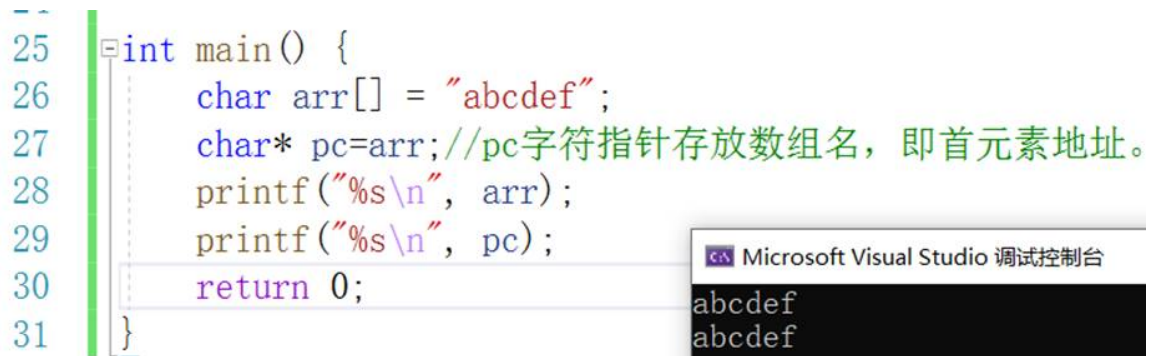
举个例子：

```
int main() {  
    char arr[] = "abcdef"; //字符串存入arr数组里面  
    char* pc=arr; //pc字符指针存放数组名，即首元素地址。  
    printf("%s\n", arr);  
    printf("%s\n", pc);  
    return 0;  
}
```

上面代码打印结果是多少呢？

因为将arr存给pc了，所以**打印结果都是abcdef**。

看一下输出结果：



The screenshot shows a code editor with the following C code:

```
25 int main() {  
26     char arr[] = "abcdef";  
27     char* pc=arr; //pc字符指针存放数组名，即首元素地址。  
28     printf("%s\n", arr);  
29     printf("%s\n", pc);  
30     return 0;  
31 }
```

To the right of the code, a window titled "Microsoft Visual Studio 调试控制台" (Microsoft Visual Studio Debug Console) displays the output of the program:

```
abcdef  
abcdef
```

## 3.内存布局

还是用上面的代码；

```
char* p="abcdef";
```

这行代码的意思，来探讨一下：

①将字符串"abcdef"存放在内存中某个位置。

"abcdef"是常量字符串。

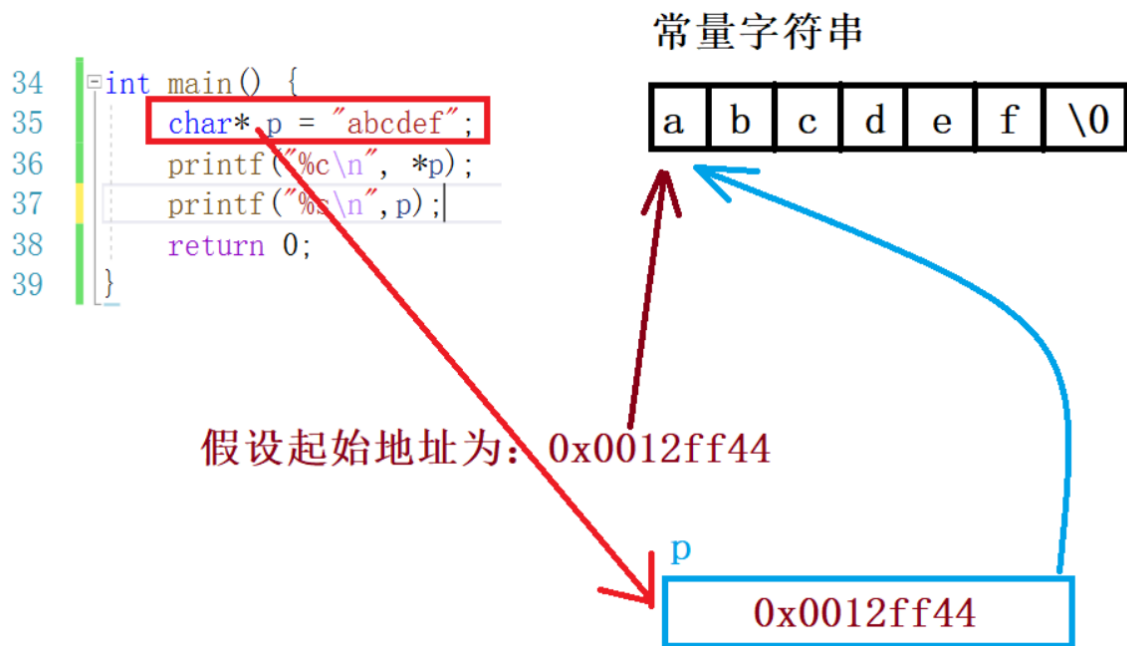
既然"abcdef"字符串放在内存中，就会有它的起始地址。

②假设它的起始地址是：0x0012ff44，

那么字符指针变量p里面存放的就是该字符串的首字符地址，即0x0012ff44。

p能够通过该地址，找到该字符串。

如下图：



③这时候将p打印出来

```
printf("%s\n", p);
```

遇到 `\0` 就停止打印，所以就可以打印出来整个字符串。

#### 补充

有的小伙伴可能不太理解为何打印整个字符串，这里解释一下：

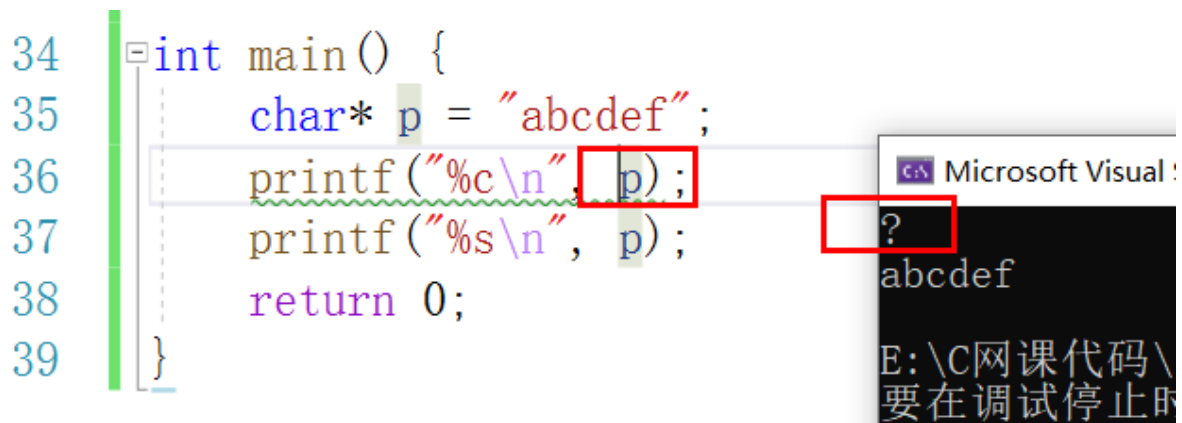
- 打印一个字符，用 `%c`，p里面存的是a的地址，`*p`就是a。
- 打印整个字符，遇到“`\0`”停止，用 `%s`，p里面存的就是a的地址。直接把p放在后面，就从p存的地址处开始打印一个字符串，就能打印出“abcdef”。

给大家看一下：

①这里要用 `*p`：



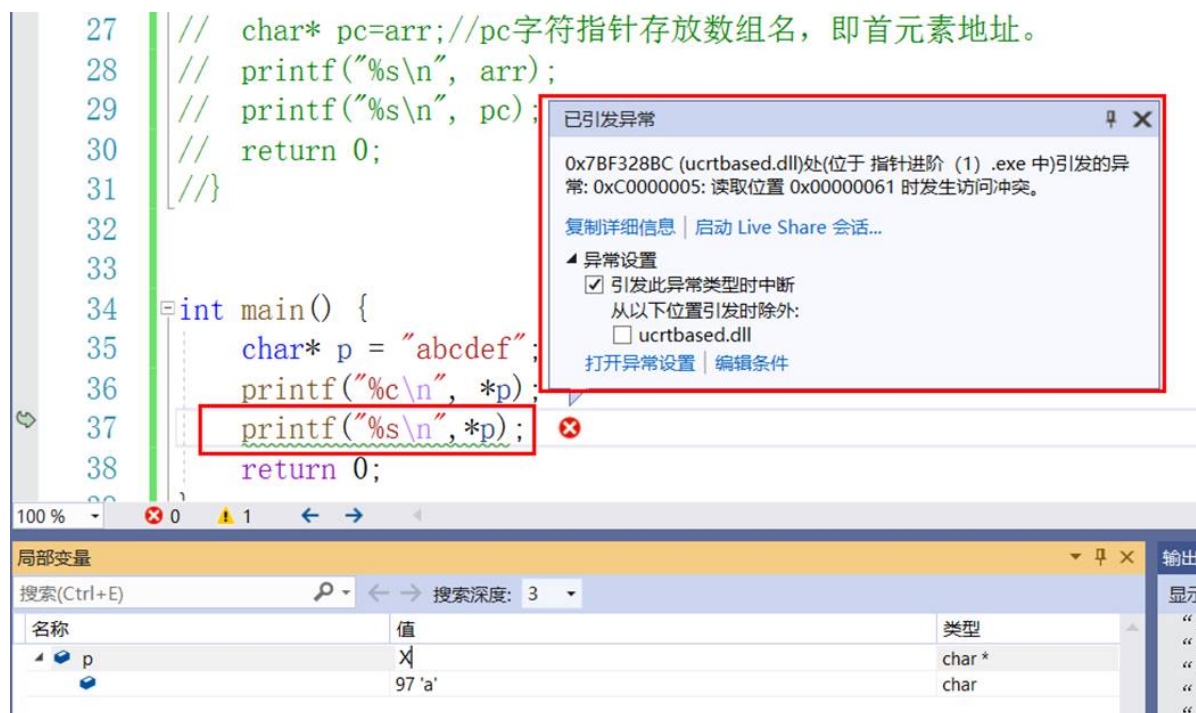
```
34 int main() {
35     char* p = "abcdef";
36     printf("%c\n", p);
37     printf("%s\n", p);
38     return 0;
39 }
```



代码	说明
 C6274	传递了非字符型作为 _Param_(2), 而对“printf”的调用需要一个字符, 实际类型: “char *”。

②这里要用 p:

```
27 // char* pc=arr;//pc字符指针存放数组名，即首元素地址。
28 // printf("%s\n", arr);
29 // printf("%s\n", pc);
30 // return 0;
31 //}
32
33
34 int main() {
35     char* p = "abcdef";
36     printf("%c\n", *p);
37     printf("%s\n", *p);
38     return 0;
39 }
```



已引发异常

0x7BF328BC (ucrtbased.dll)处(位于 指针进阶 (1) .exe 中)引发的异常: 0xC0000005: 读取位置 0x00000061 时发生访问冲突。

[复制详细信息](#) | [启动 Live Share 会话...](#)

异常设置

- ☒ 引发此异常类型时中断
- 从以下位置引发时除外:
  - ☐ ucrtbased.dll

[打开异常设置](#) | [编辑条件](#)

局部变量

名称	值	类型
p	x	char *
	97 'a'	char

最后再强调一下，

字符串要赋给指针变量p，不是把字符串的内容赋给p，而是把这个字符串的首字母的地址 赋给了p。



```
34 int main() {
35     char* p = "abcdef";
36     printf("%c\n", *p);
37     printf("%s\n", p);
38     return 0;
39 }
```

## 三、字符指针与字符串数组

可能上边大家会有点晕，这里梳理一下字符指针与字符串数组。

### 1. 字符指针

还是这行代码：

```
char* p="abcdef";
```

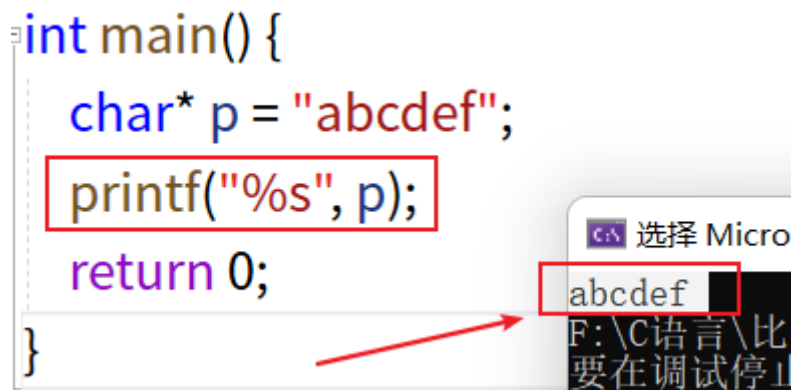
之前我们说过，这里的 "abcdef" 是**常量字符串**，常量字符串有什么需要注意的地方呢？

- 可以通过字符指针输出字符串

```
int main(){
    char* p="abcdef";
    printf("%s",p);
    return 0;
}
```

输出结果：

```
int main() {
    char* p = "abcdef";
    printf("%s", p);
    return 0;
}
```



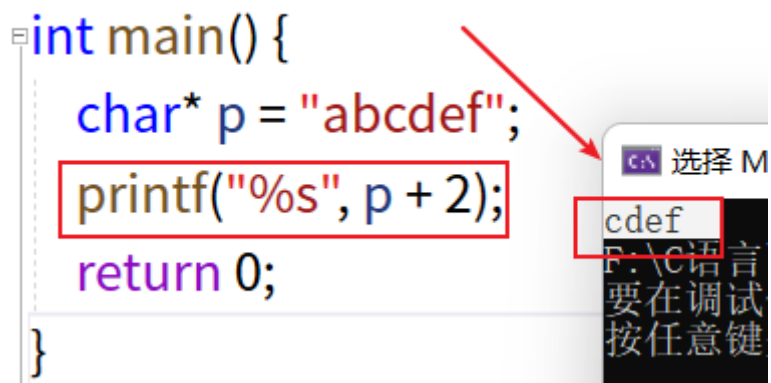
The screenshot shows a C program in a code editor. The code defines a character pointer `p` pointing to the string "abcdef" and uses `printf` to print it. A red box highlights the `printf` statement. To the right, a terminal window shows the output "abcdef" in a red box, with a red arrow pointing from the code to the output. The terminal also shows a prompt "C:\> 选择 Micro" and a file path "F:\C语言\比" with the instruction "要在调试停1".

也可以输出部分字符串:

```
int main(){
    char* p="abcdef";
    printf("%s",p+2);
    return 0;
}
```

输出结果:

```
int main() {
    char* p = "abcdef";
    printf("%s", p + 2);
    return 0;
}
```



The screenshot shows a C program in a code editor. The code defines a character pointer `p` pointing to the string "abcdef" and uses `printf` to print the substring starting from the third character (`p+2`). A red box highlights the `printf` statement. To the right, a terminal window shows the output "cdef" in a red box, with a red arrow pointing from the code to the output. The terminal also shows a prompt "C:\> 选择 M" and a file path "F:\C语言" with the instruction "要在调试按任意键".

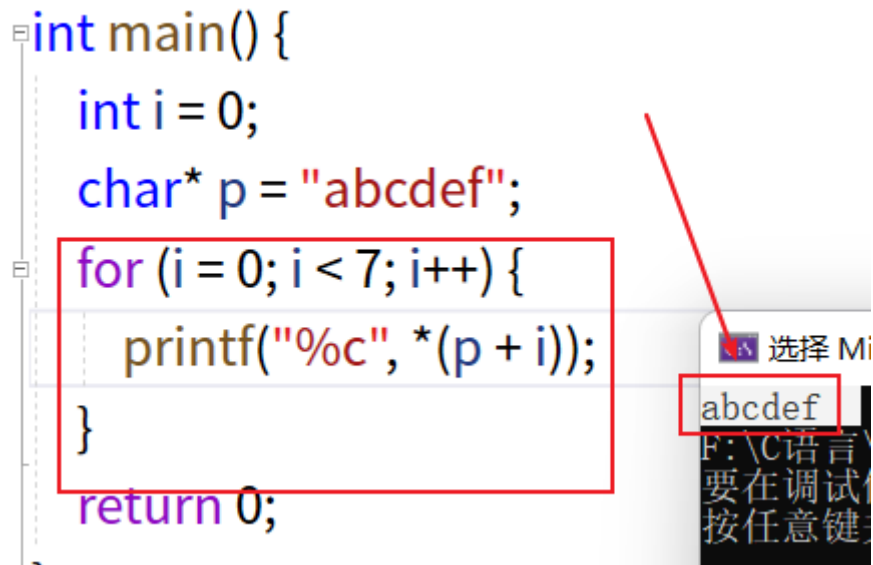
- 字符指针里面存放的是常量字符串首字符的地址，可以通过地址找到字符串每个字符。

比如我们可以输出看一下:

```
int main(){
    int i=0;
    char* p="abcdef";
    for(i=0;i<7;i++){
        printf("%c",*(p+i));
    }
    return 0;
}
```

输出结果:

```
int main() {
    int i = 0;
    char* p = "abcdef";
    for (i = 0; i < 7; i++) {
        printf("%c", *(p + i));
    }
    return 0;
}
```



- 不可以通过指针，改变常量字符串的任意字符。

既然p里面存放的是a的地址，那么 \*p 就是a。

现在想把a改成w，这样写可以吗？

```
*p = 'w';
```

输出看一下：

```
42 int main() {
43     char* p = "abcdef";
44     *p = 'w';
45     printf("%s\n", p);
46
47     return 0;
48 }
```

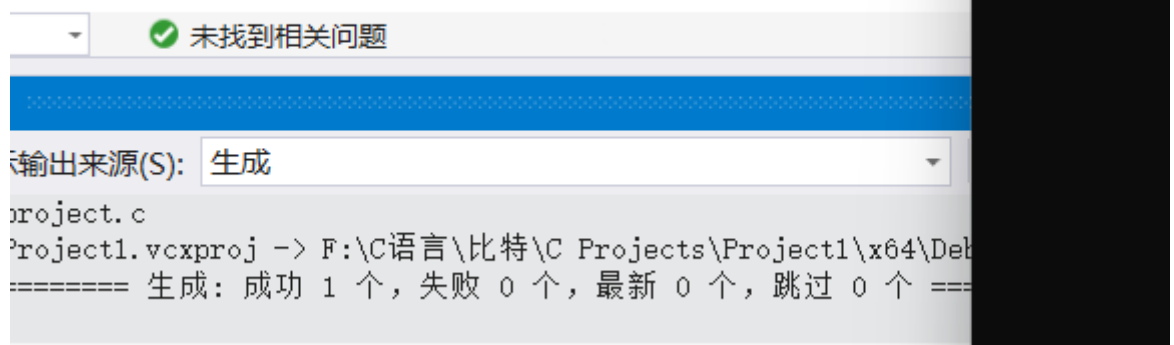
我们会发现，编译是没有问题的，但是运行是有问题的。

编译器崩溃：

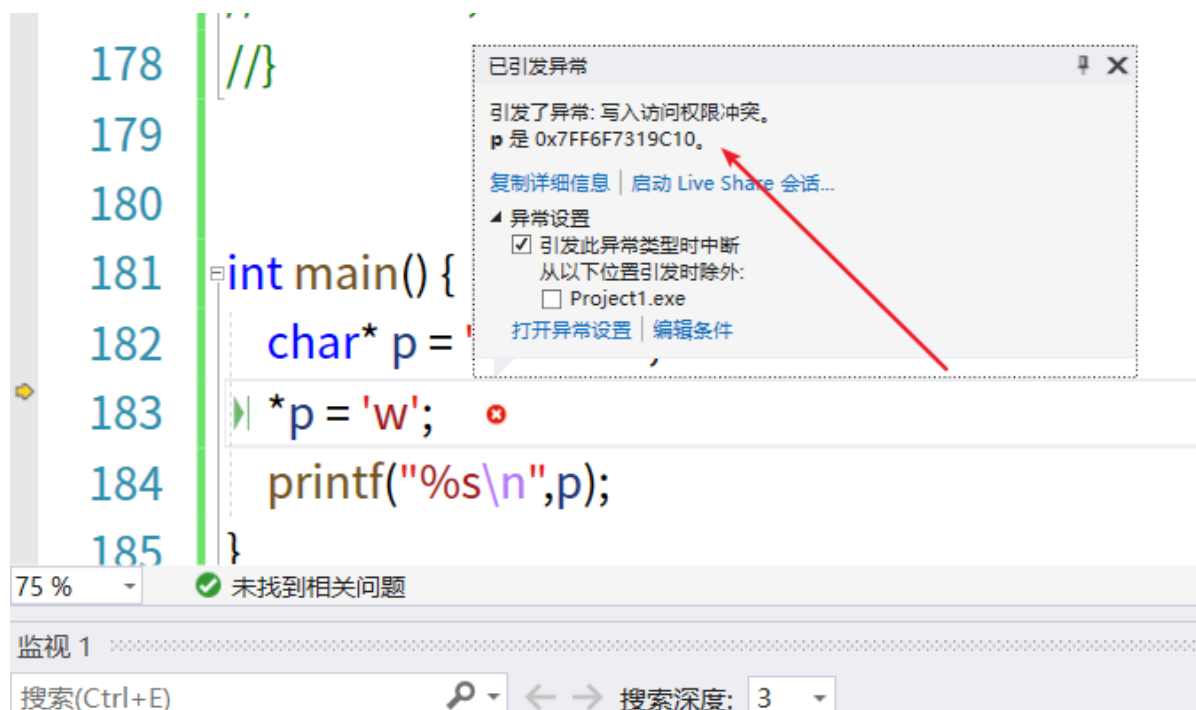
```

181 int main() {
182     char* p = "abcdef";
183     *p = 'w';
184     printf("%s\n", p);
185 }

```



调试看一下，出现异常：



如果在Linux系统下会出现 `Segmentation fault` (段错误) 的错误。

写入访问权限冲突，访问非法内存。

当时在栈溢出的时候介绍了一个网站: [www.stackoverflow.com](http://www.stackoverflow.com)

今天再介绍一个网站: [SegmentFault 思否](http://SegmentFault.com)

再回到刚才报错的代码:

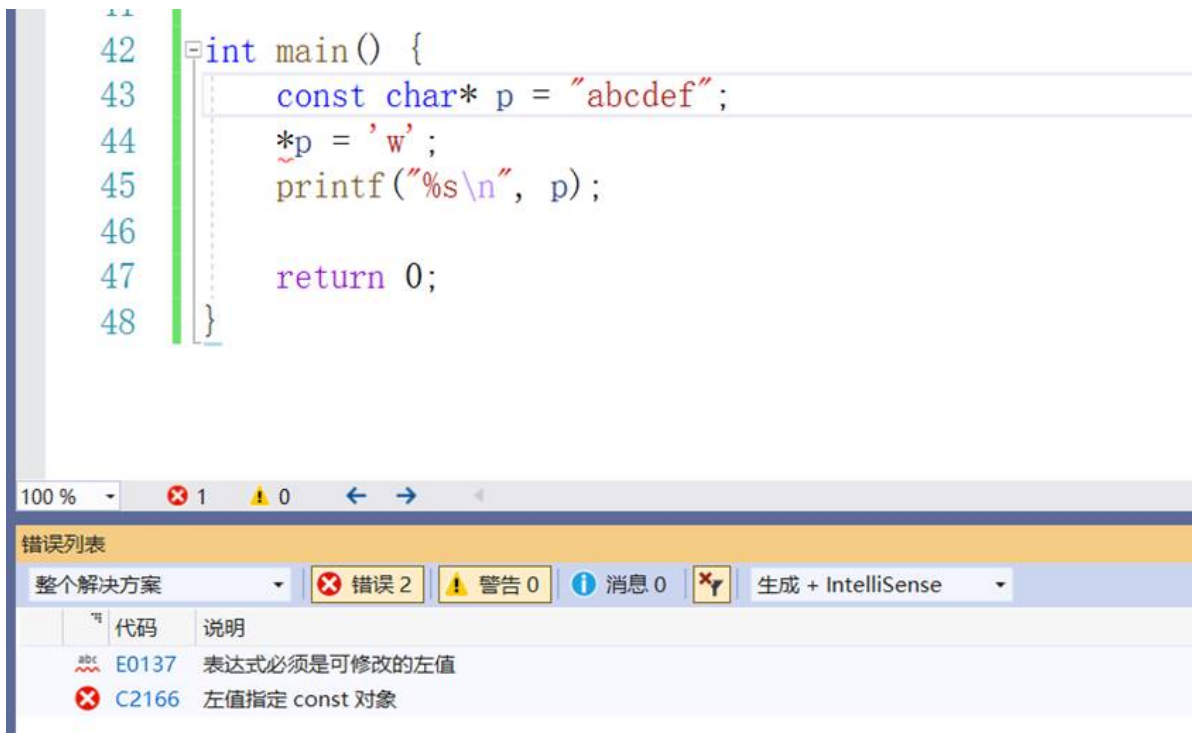
“abcdef”是一个常量字符串, 常量字符串里面的东西不能被修改!

这里正确的写法, 是需要给 `char* p` 之前加一个 `const`。

`const`修饰的是 `*p`, 即`p`所指向的内容不能被修改, `P`指向的字符串不能被修改。

```
const char* p="abcdef";
```

如果要修改, 就会报错:



```
42 int main() {
43     const char* p = "abcdef";
44     *p = 'w';
45     printf("%s\n", p);
46
47     return 0;
48 }
```

错误列表

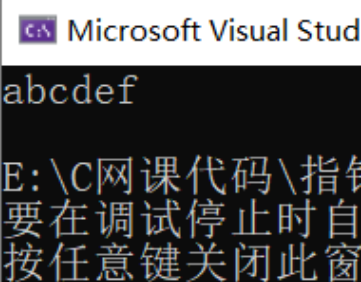
代码	说明
E0137	表达式必须是可修改的左值
C2166	左值指定 const 对象

不修改, 可以正常输出:

```

42 int main() {
43     const char* p = "abcdef";
44     /**p = 'w' ;
45     printf("%s\n", p);
46
47     return 0;
48 }

```



- 可以改变指针指向的字符串

这儿的“abcdef”是字符串常量，指针指向了放在只读内存区中的常量，只读区域不允许改变。

而w是字符常量，要是是一个字符串，就可以。

字符串数组，赋初值的时候，系统就分配好了内存，这儿的内存存在只读区域中。

如果想要修改，只能这样：（让p指针指向其他字符串）

```

int main(){
    const char* p="abcdef";
    printf("%s\n",p);
    p="abc";//让p指针指向其他字符串
    printf("%s\n",p);
    return 0;
}

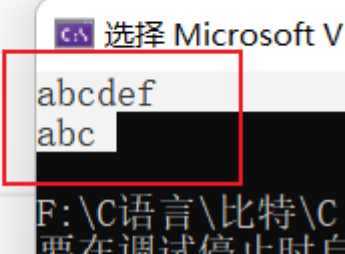
```

看一下：

```

int main() {
    const char* p = "abcdef";
    printf("%s\n", p);
    p = "abc";
    printf("%s\n", p);
    return 0;
}

```



修改后的指针p并没有修改“abcdef”的值，而是指向了一个新的字符串。

## 2.字符串数组

- 先输出看一下arr里面存的字符串。

```
int main(){
    char arr[]="abcdef";
    printf("arr=%s\n",arr);
    return 0;
}
```

输出看一下：

```
int main() {
    char arr[] = "abcdef";
    printf("arr=%s\n", arr);
    // printf("arr[0]=%c\n", arr[0]);
    return 0;
}
```




Microsoft Visual Stud  
arr=abcdef  
F:\C语言\... Pro  
要在调试停止时自动关  
按任意键关闭此窗口

可以输出整个字符串，也可以输出部分，比如：

```
int main(){
    char arr[]="abcdef";
    printf("%s\n",arr+2);
    return 0;
}
```

输出结果：

```
int main() {
    char arr[] = "abcdef";
    printf("%s\n", arr + 2);
    return 0;
}
```



选择 N  
cdef

- 常量字符串存放在数组，内存中是一个字符一个字符的存进去的。

比如“abcdef”存进去就是“abcdef\0”。

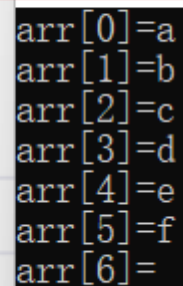


可以分别输出看一下：

```
int main() {  
    int i = 0;  
    char arr[] = "abcdef";  
    for (i = 0; i < 7; i++) {  
        printf("arr[%d]=%c\n", i, arr[i]);  
    }  
    return 0;  
}
```

输出结果：

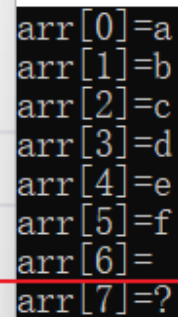
```
int main() {  
    int i = 0;  
    char arr[] = "abcdef";  
    for (i = 0; i < 7; i++) {  
        printf("arr[%d]=%c\n", i, arr[i]);  
    }  
}
```



```
arr[0]=a  
arr[1]=b  
arr[2]=c  
arr[3]=d  
arr[4]=e  
arr[5]=f  
arr[6]=
```

注意，arr[6]的值是0，如果没有值输出，就会输出问号，这个地方没有显示而已：

```
int main() {  
    int i = 0;  
    char arr[] = "abcdef";  
    for (i = 0; i < 8; i++) {  
        printf("arr[%d]=%c\n", i, arr[i]);  
    }  
}
```



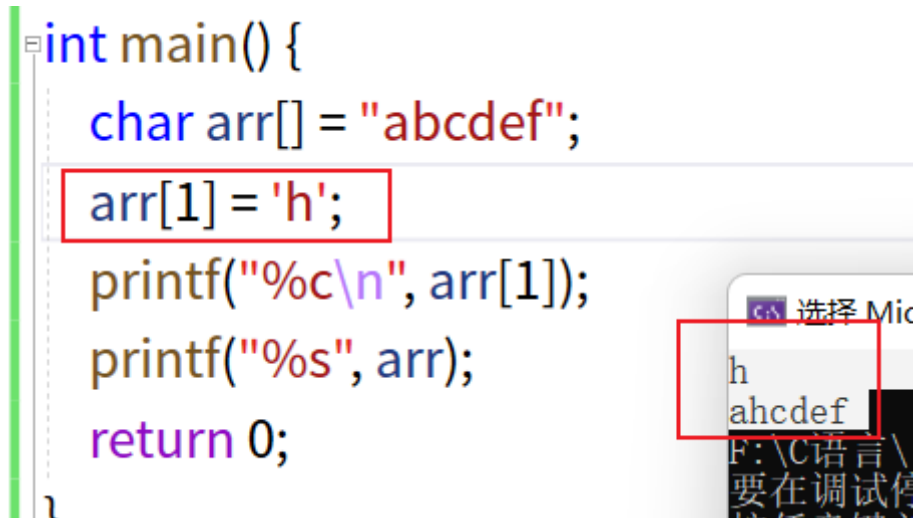
```
arr[0]=a  
arr[1]=b  
arr[2]=c  
arr[3]=d  
arr[4]=e  
arr[5]=f  
arr[6]=  
arr[7]=?
```

- 字符串既然是一个一个存放在数组，就可以修改字符串里面的字符。

比如，现在想修改第二个字符为 h。

```
int main() {
    char arr[] = "abcdef";
    arr[1] = 'h';
    printf("%c\n", arr[1]);
    printf("%s", arr);
    return 0;
}
```

输出:



```
int main() {
    char arr[] = "abcdef";
    arr[1] = 'h';
    printf("%c\n", arr[1]);
    printf("%s", arr);
    return 0;
}
```

h  
ahcdef

但是不能修改整个字符串:

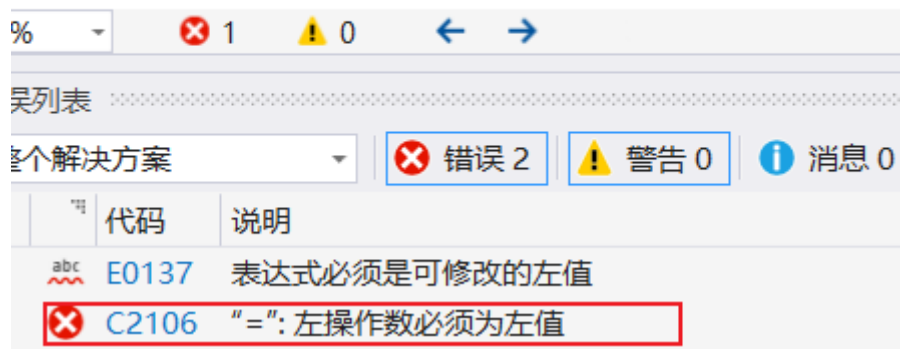
```
int main(){
    char arr[]="abcdef";
    arr="ghty";
    printf("arr=%s\n",arr);
    return 0;
}
```

输出报错:

```

495 int main() {
496     char arr[] = "abcdef";
497     arr = "ghty";
498     printf("arr=%s\n", arr);
499     return 0;
500 }

```



## 四、面试题

### 1.One

有这样一道面试题：

```

int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    char* p1 = "abcdef";
    char* p2 = "abcdef";
    if (arr1==arr2) {
        printf("hehe\n");
    }
    else {
        printf("haha\n");
    }
    return 0;
}

```

这个面试题输出结果是多少呢？

看一下：

```

49
50 int main() {
51     char arr1[] = "abcdef";
52     char arr2[] = "abcdef";
53     char* p1 = "abcdef";
54     char* p2 = "abcdef";
55     if (arr1==arr2) {
56         printf("hehe\n");
57     }
58     else {
59         printf("haha\n");
60     }
61     return 0;
62 }

```

Microsoft V

haha

E:\C网 课代  
要在调试停  
按任意键关

❓ 为什么结果是“haha”？

我们创建了两个数组，`arr1` 和 `arr2`，它们在内存中一定是有两块空间的，两个数组的数组名当然是两个不同的首元素地址了。

即`arr1`不等于`arr2`的地址。所以打印“haha”。

## 2.Two

我们再换一个代码看看：（注意if语句里面的判断）

```

int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    char* p1 = "abcdef";
    char* p2 = "abcdef";
    if (p1 == p2) {
        printf("hehe\n");
    }
    else {
        printf("haha\n");
    }
    return 0;
}

```

打印输出看一下：

The screenshot shows a C program in a debugger window titled 'test.c'. The code defines two character arrays, 'arr1' and 'arr2', both containing the string 'abcdef'. It also defines two pointers, 'p1' and 'p2', both pointing to the string 'abcdef'. The program has an 'if' statement that checks if 'p1 == p2'. If true, it prints 'hehe\n'; otherwise, it prints 'haha\n'. The debugger shows the program has executed the 'if' statement and printed 'hehe\n'. A console window on the right shows the output 'hehe'.

```
test.c 指针进阶 (1) (全局范围)
50 int main() {
51     char arr1[] = "abcdef";
52     char arr2[] = "abcdef";
53     char* p1 = "abcdef";
54     char* p2 = "abcdef";
55     if (p1 == p2) {
56         printf("hehe\n");
57     }
58     else {
59         printf("haha\n");
60     }
61     /*if (arr1==arr2) {
62         printf("hehe\n");
63     }
64     else {
65         printf("haha\n");
66     }*/
67 }
```

? 现在打印的结果为啥是“hehe”?

常量字符串 不能被修改。

两个字符串既然一模一样，又是常量字符串，各自又不能修改，没有必要再在内存中存两份。只需要拿去用，不能改它。

为了在内存中节省空间，这两个“abcdef”只存了一份。（这一点非常重要）

不管是p1还是p2，都指向同一块空间的起始位置。

而第一次没有修改代码之前，是创建两个不同的数组，地址是不一样的。

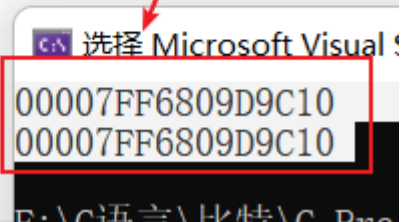
### 3.探究

①不妨输出看一下p1与p2指向的地址，看一下是否相同：

```
int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    char* p1 = "abcdef";
    char* p2 = "abcdef";
    printf("%p\n", p1);
    printf("%p\n", p2);
    return 0;
}
```

看一下结果，是一样的：（注意是它们指向的地址相同，这两个指针本身的地址是不一样的）

```
188 int main() {
189     char arr1[] = "abcdef";
190     char arr2[] = "abcdef";
191     char* p1 = "abcdef";
192     char* p2 = "abcdef";
193     printf("%p\n", p1);
194     printf("%p\n", p2);
195     return 0;
```

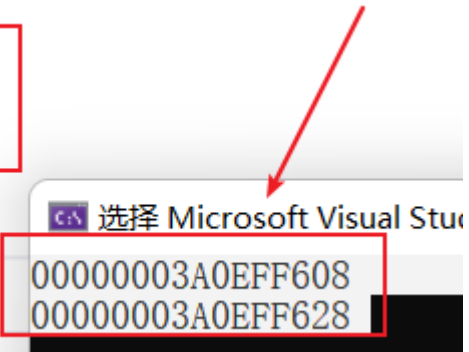


②再看一下p1与p2本身的地址：

```
int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    char* p1 = "abcdef";
    char* p2 = "abcdef";
    printf("%p\n", &p1);
    printf("%p\n", &p2);
    return 0;
}
```

输出看一下，结果是不一样的：（两个指针的本身地址不同）

```
int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    char* p1 = "abcdef";
    char* p2 = "abcdef";
    printf("%p\n", &p1);
    printf("%p\n", &p2);
    return 0;
}
```



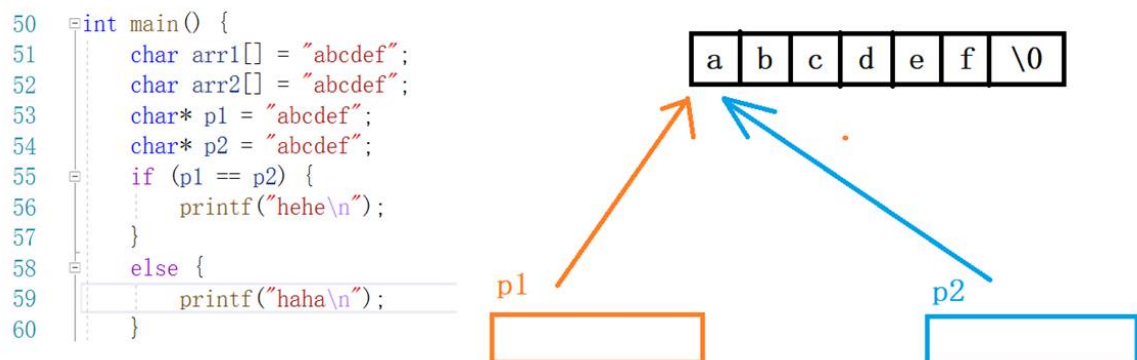
这里也能发现，if语句判断的是，两个指针指向的空间地址是否相同。而不是判断两个指针本身的地址。

## 4.补充

**改变p1并不能改变p2。**

更何况p1和p2指向的常量字符串是不可修改的。

p1和p2是两个独立的空间，指向同一个地址：



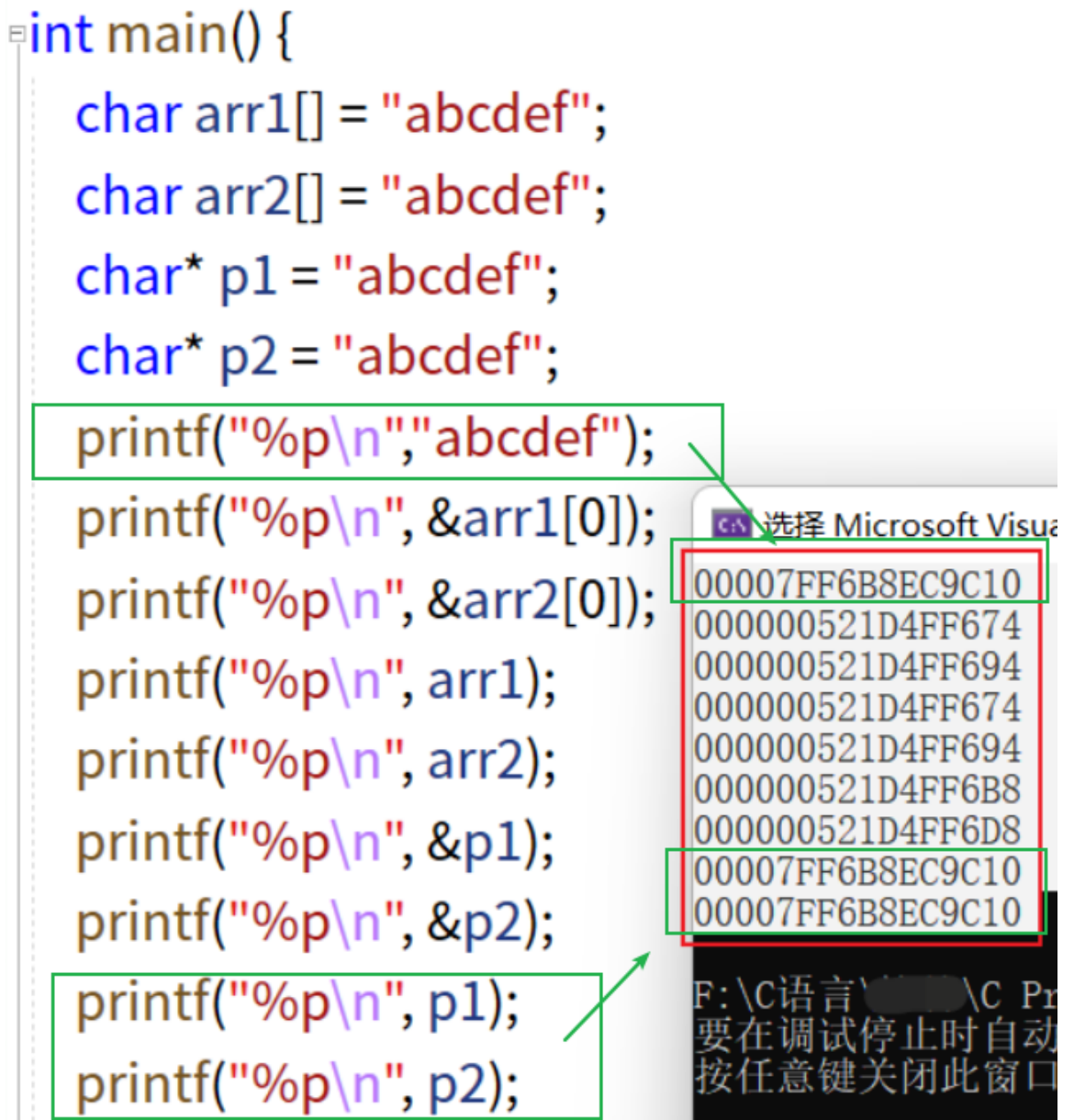
**标准写法：**

标准写法，需要给字符指针前面加上 `const`：

```
int main() {
    char arr1[] = "abcdef";
    char arr2[] = "abcdef";
    const char* p1 = "abcdef";
    const char* p2 = "abcdef";
    return 0;
}
```

p1与p2指向的地址就是字符串的地址，如下：





## 五、地址问题

关于上面的面试题，可能还有小伙伴不明白。

这里将它们的地址都打印出来，一起做个比较。

测试一段代码：

```
int main() {  
    char arr1[] = "abcdef";  
    char arr2[] = "abcdef";  
    char* p1 = "abcdef";  
    char* p2 = "abcdef";  
    printf("abcdef=%p\n", "abcdef");  
  
    printf("&arr1[0]=%p\n", &arr1[0]);  
    printf("&arr2[0]=%p\n", &arr2[0]);  
    printf("arr1=%p\n", arr1);  
}
```

```

printf("arr2=%p\n", arr2);

printf(" &p1=%p\n", &p1);
printf("&p2=%p\n", &p2);
printf("p1=%p\n", p1);
printf("p2=%p\n", p2);
printf("&(*p1)=%p\n", &(*p1));
printf("&(*p2)=%p\n", &(*p2));
return 0;
}

```

输出结果：

```

abcdef=00007FF705AE9CE8
&arr1[0]=0000007F680FF7C4
&arr2[0]=0000007F680FF7E4
arr1=0000007F680FF7C4
arr2=0000007F680FF7E4
 &p1=0000007F680FF808
&p2=0000007F680FF828
p1=00007FF705AE9CE8
p2=00007FF705AE9CE8
&(*p1)=00007FF705AE9CE8
&(*p2)=00007FF705AE9CE8

```

可以看到，地址大小：

```

"abcdef"==p1==p2==&(*p1)==&(*p2)

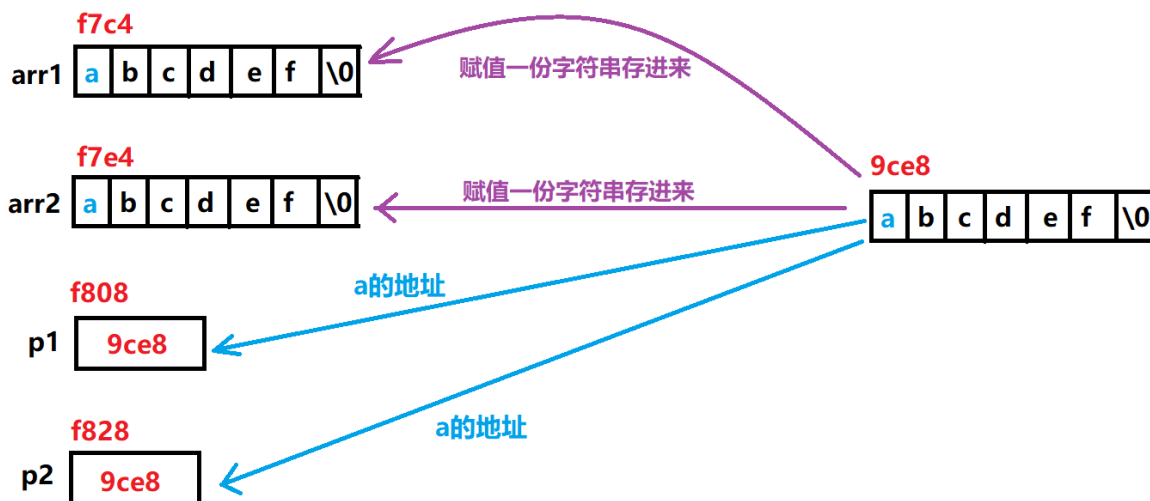
&arr1[0]==arr1

&arr2[0]==arr2

&p1!=&p2

```

内存图：



## 六、字符数组与字符串数组

上边写到了字符串数组，这里提一嘴字符数组与字符串数组。

和指针无关，可自行跳过。

### 1.sizeof与strlen

#### 含义

`sizeof()` 是运算符，在头文件的类型为 `unsigned int`，其运算值在编译时就计算好了，参数可以是**指针、数组、类型、对象和函数**等；

`strlen()` 是函数，要在运行时才能计算。参数必须是**字符型指针** (`char*`)。当数组名作为参数传入时，实际上数组就退化为指针了。该函数完成的功能是从代表该字符串的第一个地址开始遍历的，直到遇到结束符 `NULL`。返回的长度大小不包括 `NULL`。

#### 示例一

```
#include <stdio.h>
#include <string.h> //strlen要引头文件

int main()
{
    char str[20] = "hello";
    printf("strlen=%d\n", strlen(str));
    printf("sizeof=%d\n", sizeof(str));
    return 0;
}
```

结果显示为：

```
strlen = 5
sizeof = 20
```

这时的 `strlen`=5, `sizeof`=20

因为 `strlen` 计算的是字符串的长度，以 `\0` 为字符串结束标志；

而 `sizeof` 计算的是分配的数组 `str[20]` 所占的内存空间的大小，不受里面存储的内容影响。

## 示例二

```
#include <stdio.h>

int main()
{
    char *str1 = "abcde";
    char str2[] = "abcde";
    char str3[8] = {'a'};
    char str4[] = "0123456789";

    printf("sizeof(str1)=%d\n", sizeof(*str1));
    printf("sizeof(str2)=%d\n", sizeof(str2));
    printf("sizeof(str3)=%d\n", sizeof(str3));
    printf("sizeof(str4)=%d\n", sizeof(str4));

    return 0;
}
```

结果显示为：

```
sizeof(str1) = 4
sizeof(str2) = 6
sizeof(str3) = 8
sizeof(str4) = 11
```

`str1` 是一个指针，只是指向了字符串 `"abcde"` 而已。所以 `sizeof(*str1)` 不是字符串占的空间，也不是字符数组占的空间，而是一个**指针所占的空间**。在C/C++中**一个指针占四个字节**。（32平台）

`str2` 是一个字符型数组，对于一个数组，返回这个数组所占的总空间，所以 `sizeof(str2)` 取得的是字符串 `"abcde"` 的总空间。`"abcde"` 中，共有 `a b c d e \0` 六个字符，所以 `str2` 数组的长度为6。

`str3` 已经定义成了长度为8的数组，所以 `sizeof(str3)` 为8；

`str4` 和 `str2` 类似，共十一个字符，所以 `str4` 所占的空间是11。

## 说明

示例二里面列举了一个指针，如果你验证我的代码的话，可能会是 `sizeof(str1) = 8`

那是因为32位机器上**指针大小是4个字节，64位机器上是8个字节**

因为**32位机器的寻址地址空间是4G**，每个地址是32位，恰好是4个字节。即指针大小是4个字节。

而**64位机器的每个地址是64位**，是8个字节，因此指针是8个字节。

## 代码三

子函数中，`sizeof` 会把从主函数中传进来的字符数组当作是指针来处理。

指针的大小又是由机器来决定，而不是人为的来决定的。

```
void size_of(char str[])
{
    printf("sizeof = %d\n", sizeof(str));
}

int main()
{
    char str[20] = "hello";
    size_of(str);
    return 0;
}
```

结果显示为：

```
sizeof = 4
```

具体而言，当参数分别是如下时，`sizeof`返回的值表示的含义如下：

数组：编译时分配的数组空间的大小；

指针：存储该指针所用的空间的大小（存储该指针的地址的长度，是长整型，应该是4）；

类型：该类型所占的空间的大小；

对象：对象的实际占用空间大小；

函数：函数的返回类型所占的空间大小。函数的返回类型不能是void。

## 代码四

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

int main()
{
    char *str = "0123456789";
    printf("sizeof(str) = %d\n", sizeof(str));
    printf("sizeof(*str) = %d\n", sizeof(*str));
    printf("strlen(str) = %d\n", strlen(str));
    return 0;
}
```

结果显示为：

```
sizeof(str) = 4
sizeof(*str) = 1
strlen(str) = 10
```

`sizeof(str)`：`str` 是指向是字符串常量的字符指针，`sizeof` 获得的是第一个指针所占的空间，应该是长整型，所以是4；

`sizeof(*str)`：`*str` 是第一个字符，其实就是获得了字符串的第一位 `'0'` 所占的内存空间，是char类型的，占了1位。

`strlen(str)`：如果要获得这个字符串的长度，则一定要用 `strlen`。

## 总结

`sizeof` 是运算符，测量的是**字符的分配大小**

`strlen` 是函数，测量的是**字符的实际长度**，以 `\0` 结束，所以只要 `strlen` 碰到 `\0` 就结束

## 2. 字符数组与字符串数组

讲之前，我们还是先回顾一下关于 `sizeof` 和 `strlen` 的用法

`strlen`：

- 是一个库函数；
- 计算的是字符串的长度，并且只针对字符串；
- 关注的字符串中是否有 `\0`，计算的是 `\0` 之前的字符个数；

`sizeof`：

- 是一个操作符（运算符）；
- `sizeof` 是用来计算变量所占内存空间大小的，任何类型都可以使用；
- 只关注空间大小，不在乎内存中是否存在 `\0`；

### 示例一

`' '`：表示一个字符；

`" "`：表示一个字符串；

- `arr1`里面的元素是字符，表示用字符初始化字符数组；
- `arr2`里面的元素是字符串，表示用字符串初始化字符数组；

```
#include <stdio.h>

int main()
{
    char arr1[] = { 'a', 'b', 'c' };
    printf("%d\n", sizeof(arr1));

    char arr2[] = { "abc" };
    printf("%d\n", sizeof(arr2));

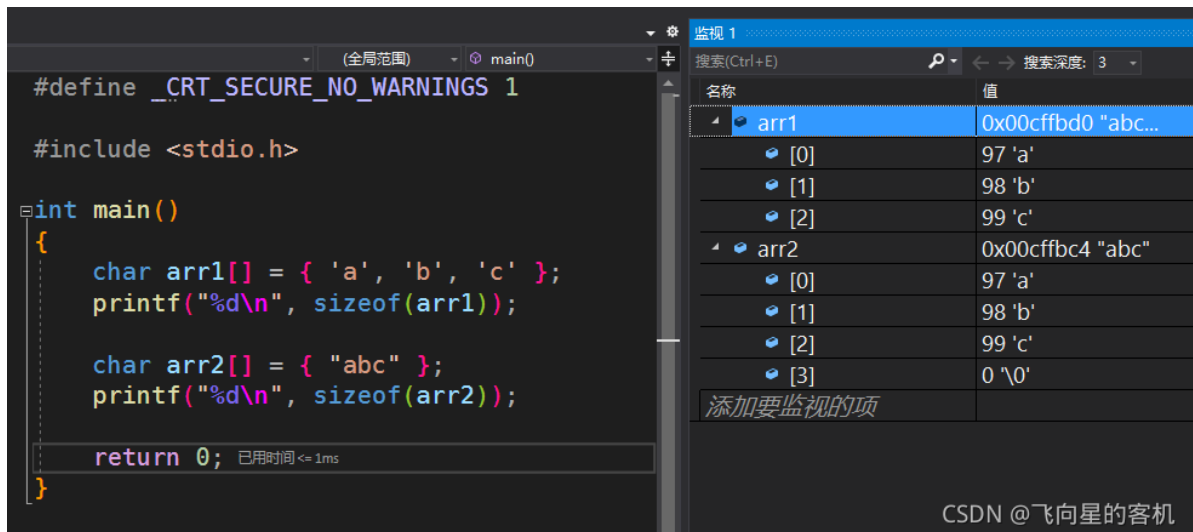
    return 0;
}
```

123456789101112

运行结果：

3  
4  
12

解析:



`sizeof` 打印的是所占空间的大小

`arr1`里面的元素是: 'a'、'b'、'c', 所以大小就是3个字节

`arr2`里面的元素是: a、b、c、\0, 所以大小就是4个字节

## 示例二

```
#include <stdio.h>

int main()
{
    char arr1[] = { 'a', 'b', 'c' };
    printf("%d\n", strlen(arr1));

    char arr2[] = { "abc" };
    printf("%d\n", strlen(arr2));

    return 0;
}
```

123456789101112

运行结果:

15  
3  
12



解析:

```
#define __CRT_SECURE_NO_WARNINGS 1

#include <stdio.h>

int main()
{
    char arr1[] = { 'a', 'b', 'c' };
    printf("%d\n", strlen(arr1));

    char arr2[] = { "abc" };
    printf("%d\n", strlen(arr2)); 已用时间 <= 1ms

    return 0;
}
```

名称	值
arr1	0x00b6f9f0 "abc..."
arr1[0]	97 'a'
arr1[1]	98 'b'
arr1[2]	99 'c'
arr2	0x00b6f9e4 "abc"
arr2[0]	97 'a'
arr2[1]	98 'b'
arr2[2]	99 'c'
arr2[3]	0 '\0'
添加要监视的项	

CSDN @飞向星的客机

strlen 求字符串长度的时候, 关注的是 \0;

arr1里面没有 \0, 所以长度是未知的, 是一个随机值15;

arr2里面有 \0, 所以计算 \0 前面的, 也就是 a, b, c, 所以长度为3;

## 总结

```
char arr1[] = { 'a', 'b', 'c' };
//arr1有三个元素, 数组的大小是3个字节;
printf("%d\n", sizeof(arr1));
printf("%d\n", strlen(arr1)); //长度为随机值;

char arr2[] = { "abc" };
//arr2有四个元素, 数组的大小是4个字节;
printf("%d\n", sizeof(arr2));
printf("%d\n", strlen(arr2)); //长度为3;
```

📖 参考文章:

[【C语言深度剖析】深入理解字符数组和字符串数组Albert Edison的博客-CSDN博客c语言字符数组和字符串数组](#)

(1条消息) [【C语言深度剖析】详解strlen与sizeof的区别及用法 Albert Edison的博客-CSDN博客](#)

欢迎关注, 一位喜欢慢慢生活的博主。

自述文件



雨翼轻尘

110

3.3w

2.1w

原创内容

作者排名

粉丝数量



CSDN