**1. 指针**

1.1 一个指针包含两方面：a) 地址值；b) 所指向的数据类型。

1.2 解引用操作符（dereference operator）会根据指针当前的地址值，以及所指向的数据类型，访问一块连续的内存空间（大小由指针所指向的数据类型决定），将这块空间的内容转换成相应的数据类型，并返回左值。

有时候，两个指针的值相同，但数据类型不同，解引用取到的值也是不同的，例如，

**[cpp]** [view plain](https://blog.csdn.net/wu_nan_nan/article/details/51741030) [copy](https://blog.csdn.net/wu_nan_nan/article/details/51741030)

1. **char** str[] ={0, 1, 2, 3};       /\* 以字符的ASCII码初始化 \*/
3. **char** \* pc = &str[0];        /\* pc指向str[0]，即0 \*/
5. **int** \* pi = (**int** \*) pc;      /\* 指针的“值”是个地址，32位。 \*/

此时，pc和pi同时指向str[0]，但\*pc的值为0（即，ASCII码值为0的字符）；而\*pi的值为50462976。或许把它写成十六进制会更容易理解：0x03020100（4个字节分别为3,2,1,0）。我想你已经明白了，指针pi指向的类型为int，因此在解引用时，需要访问4个字节的连续空间，并将其转换为int返回。

**2. 数组**

2.1 数组名和指针

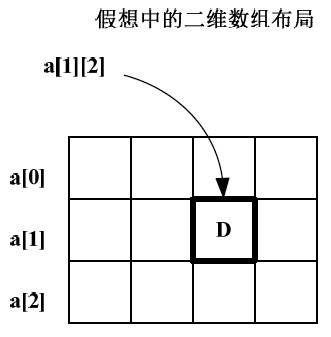
通常我们认为数组名是一个**指针常量**（例如，int a[10]; 那么a是一个int \* const），这种理解是不全面的，正确的理解如下：

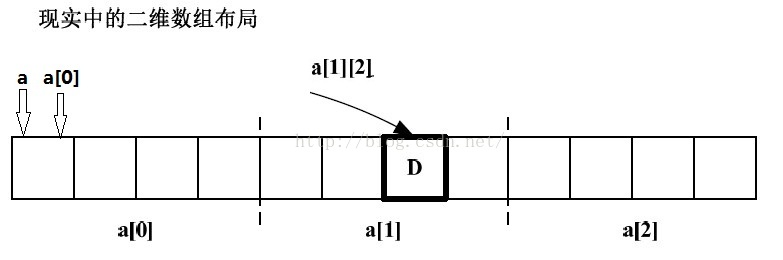
作为右值（例如，赋值语句右边）时数组名可视为指针常量（系统自动转换）；作为左值，例如取地址，sizeof，则不能视为指针。

sizeof(一个数组)返回的是数组大小\*每个元素占字节数；而sizeof(一个指针）返回4。

2.2 二维数组

实际上，不管是一维还是多维数组，都是内存中一块线性连续空间，因此在内存级别上，其实都只是一维。





（图片在参考资料2基础上稍作更改）

做如下定义：

**[cpp]** [view plain](https://blog.csdn.net/wu_nan_nan/article/details/51741030) [copy](https://blog.csdn.net/wu_nan_nan/article/details/51741030)

1. **int** a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
3. **int** \*\* p;
5. p = (**int**\*\*)a;       /\* 不做强制类型转换会报错 \*/

说明：

1）p是一个二级指针，它首先是一个指针，指向一个int\*；

2）a是二维数组名，它首先是一个指针，指向一个含有4个元素的int数组；

由此可见，a和p的类型并不相同，如果想将a赋值给p，需要强制类型转换。

**3. 为什么不能将二维数组名传递给二级指针？**

假如我们将a赋值给p，p = (int\*\*)a; 既然p是二级指针，那么当然可以这么用：\*\*p; 这样会出什么问题呢？

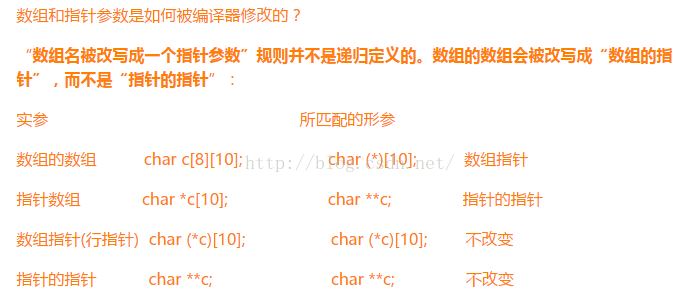
1）首先看一下p的值，p指向a[0][0]，即p的值为a[0][0]的地址；

2）再看一下\*p的值，p所指向的类型是int\*，占4字节，根据前面所讲的解引用操作符的过程：从p指向的地址开始，取连续4个字节的内容。得到的正式a[0][0]的值，即0。

3）再看一下\*\*p的值，诶，报错了？当然报错了，因为你访问了地址为0的空间，而这个空间你是没有权限访问的。

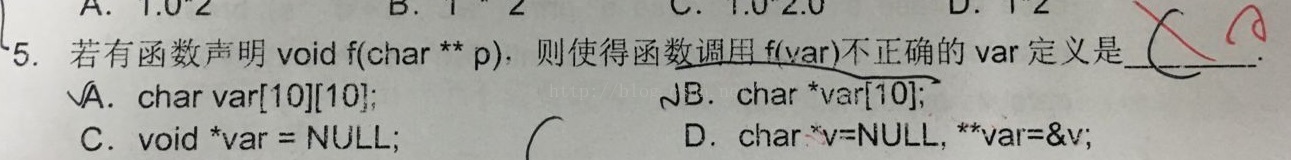
实际上这是某一年华为的面试题。感兴趣的还可以把a的类型定义为char类型的二维数组，看看会发生什么。

**4. 二维数组和二级指针相关的参数匹配**



（来源参考资料3）

**5. 那么问题来了，下题答案是？**



**方法二：通过二级指针传递<1>**

这种方法在函数内部可以使用[]符号访问数组元素，但是在调用函数前必须先开辟一段二级指针指向的内存，然后把数组值拷贝到这一块内存中，用完后必须手动释放内存。优点是同一个函数可以接收不同长和宽的二维数组，缺点是必须手动开辟一段内存用完后还要手动回收，并且要把二维数组的值拷贝到开辟的内存中。

**[cpp]** [view plain](https://blog.csdn.net/u013752202/article/details/49688717) [copy](https://blog.csdn.net/u013752202/article/details/49688717)

1. #include <QtCore/QCoreApplication>
2. #include "opencv2\opencv.hpp"
3. **void** print(**float** \*\*tab,**int** rows,**int** cols)
4. {
5. **for**(**int** i=0;i<rows;i++){
6. **for**(**int** j=0;j<cols;j++){
7. cout<<tab[i][j]<<" ";
8. }
9. cout<<endl;
10. }
11. }
12. **int** main(**int** argc, **char** \*argv[])
13. {
14. QCoreApplication a(argc, argv);
15. **float** ta[2][3]={{1.0,2.0,3.0},{4.0,5.0,6.0}};
16. **float** \*\*p=**new** **float** \*[2];//开辟行空间
17. **for**(**int** i=0;i<3;i++)
18. p[i]=**new** **float**[i];//开辟列空间
19. **for**(**int** i=0;i<2;i++){    //赋值
20. **for**(**int** j=0;j<3;j++){
21. p[i][j]=ta[i][j];
22. }
23. }
24. cout<<"ta: "<<endl;
25. print(p,2,3);//打印
26. //p的内存释放方式
27. **for**(**int** i=0;i<3;i++)
28. **delete**[]p[i];
29. **delete** []p;
30. **return** a.exec();
31. }

**766. Toeplitz Matrix 感悟**

目前该程序已有两种解法可以被网站接受，第一种是用首行和首列的元素与对应对角线上的元素比较，判定是否满足托普利兹矩阵；第二种使用每个元素与其斜对应的元素比较。相对比来说，第二种方法更省时并且易懂。

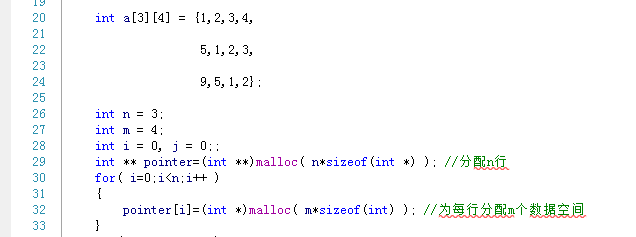
数组的名称可以表示为指针，意义并不一样，有时候，两个指针的值相同，但数据类型不同，解引用取到的值也是不同的，例如，

**[cpp]** [view plain](https://blog.csdn.net/wu_nan_nan/article/details/51741030) [copy](https://blog.csdn.net/wu_nan_nan/article/details/51741030)

1. **char** str[] ={0, 1, 2, 3};       /\* 以字符的ASCII码初始化 \*/
3. **char** \* pc = &str[0];        /\* pc指向str[0]，即0 \*/
5. **int** \* pi = (**int** \*) pc;      /\* 指针的“值”是个地址，32位。 \*/

此时，pc和pi同时指向str[0]，但\*pc的值为0（即，ASCII码值为0的字符）；而\*pi的值为50462976。或许把它写成十六进制会更容易理解：0x03020100（4个字节分别为3,2,1,0）。我想你已经明白了，指针pi指向的类型为int，因此在解引用时，需要访问4个字节的连续空间，并将其转换为int返回。

首先对于二维数组的指针并不能够单独按照指针传递的方式作为实参传入给函数，如果按照指针类型的传输的话，函数内部访问数组的参数将会出错，具体在于除了首行的成员外，其他行的变量没法访问。所以在传入二维数组的指针时，应该先将指针转换为二维数组类型，具体实现方法如下：



这段程序将指向指针的指针pointer转换为二维数组类型的指针，这样在进行参数传递后，子函数内也能识别到pointer为二维数组类型的指针。