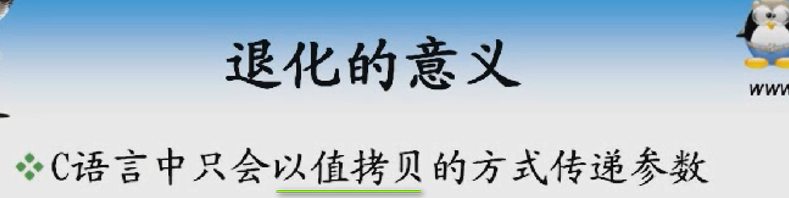
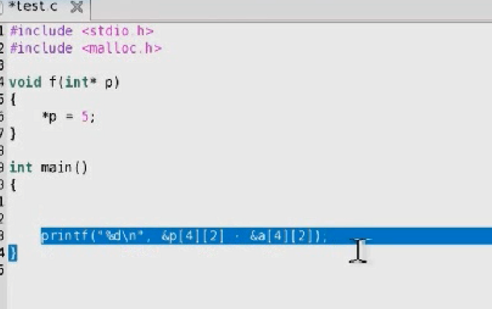
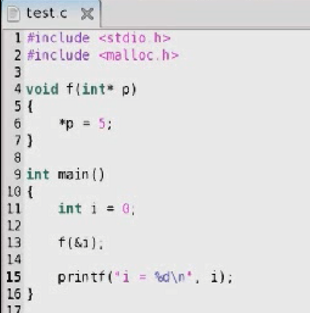
C语言的编译器 为什么必须退化为指针？

【java继承了C语言 所以 Java也是只会以传值的方式进行参数传递】

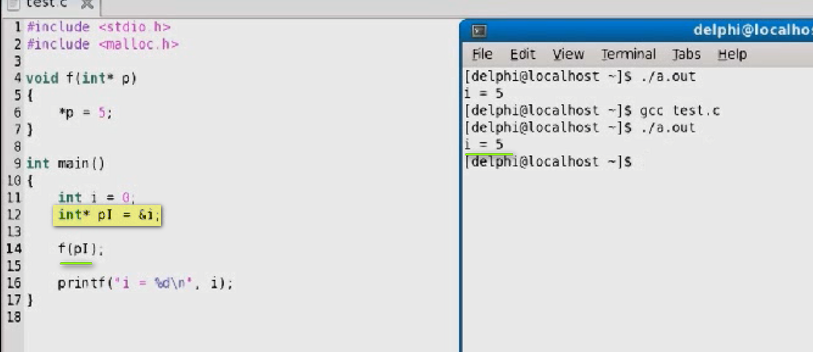
之前说的是传值 传址 --- 但是 传址也是间接的传值

 这个f函数 是传递地址 要修改函数以外的变量 【js什么的有闭包 所以 好像就不用c++这么麻烦】

 打印出5

本质上看 C语言中的函数调用 仅仅支持一种调用 就是传值

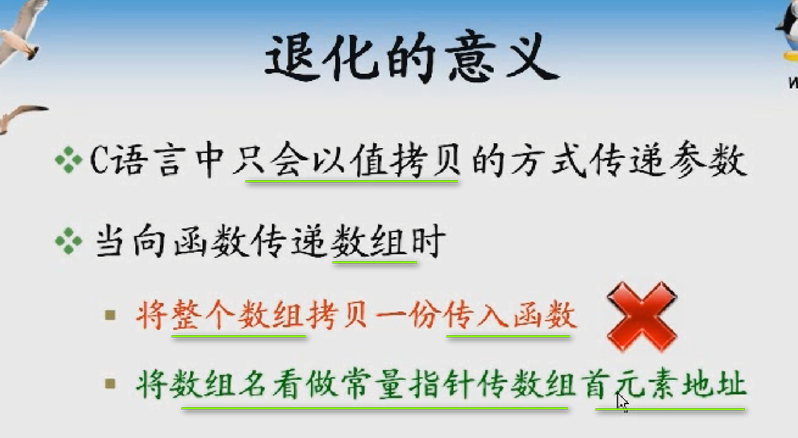
修改代码



PI这个变量的值 传给了参数p 然后 使用的是p

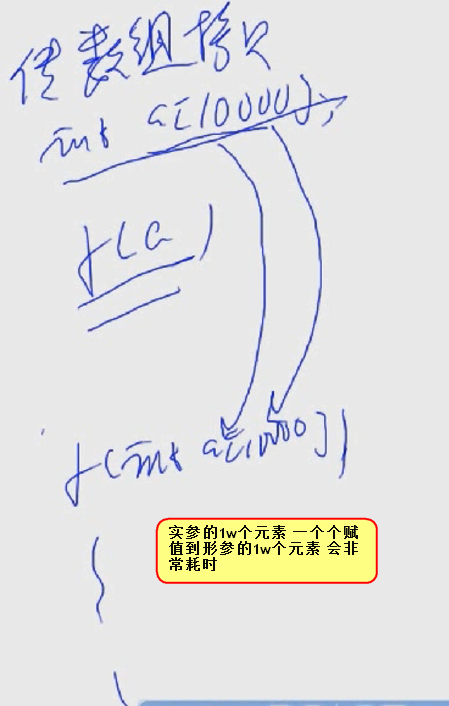
**这两种调用 ---- 实际上是把地址的值传递过去 --- C语言中仅仅有一种调用**

这样 传递数组的时候 ------ 有两种办法



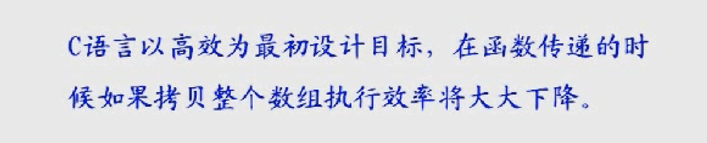
**最终C语言选择了后者 ----- 原因是什么？**

\*\*传递数组本身：如果数组很大（大不大都是未知的 所以 C不冒这种风险） ---- 就会把大的数组的参数 一个个进行copy 非常耗时

 --- 赋值4\*10000B

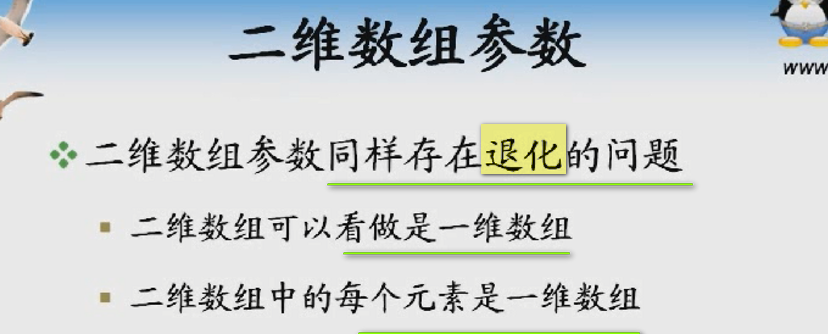
\*\*传递数组首地址：实际调用是f(a) ---- 形参是f(int a[10000]) ---但是 退化了 ------ 所以 仅仅赋值4B

为什么设计C语言？C语言设计的初期 必须高效 ----- 将整个数组copy一份传入函数 就非常低效 ----- 传递数组首元素的地址就是非常高效



**不管函数参数写成什么样的数组 最终编译器都会把数组参数退化成指针**

*----- 这个就是数组参数退化的意义 就是为了高效*

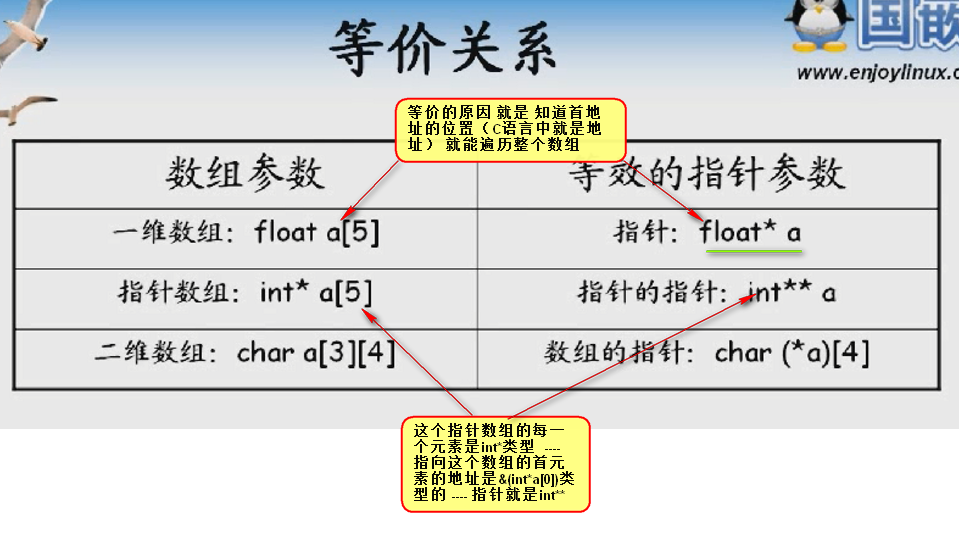


【**因为一维数组 如果使用指针进行遍历 ----- 起始位置（C语言把位置看成地址）+长度** 就可以实现指针对一维数组的遍历 ----- 所以 实参是一维数组名 形参是指向一维数组首元素的指针 ---- 也就是 我只要找到了你的一维数组的第一个元素 就基本能找到你所有的元素 ----- **所以 形参是指向一个元素的指针 ===== 此时 你给我一个数组长度 我就能使用指针获取到整个数组**

同样道理：二维数组的首元素是第一个一维数组 ----- 那么 同样道理 我的实参如果是二维数组首元素的地址 ---- 也就是二维数组的函数名 ----- 那么 我的**形参就应该是指向这个二维数组首元素的指针** ----- 所以 就退化成**指向数组的指针**就对了

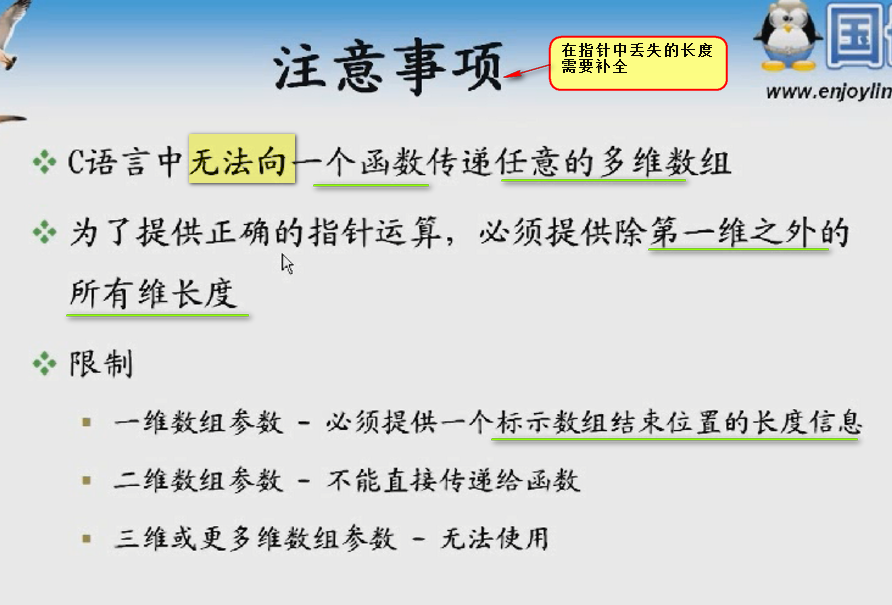
------- 指向数组的指针 ----- 以int为例 那就是 int(\*pArray)[N] ----- 含有了列的维度（因为数组 和 指向数组的指针的类型都和数组的大小是有着密切联系的 也就是 数组的大小构成了数组类型的一份子）----- 所以 第二个我只给出数组的行数就可以了 ---- 此时 利用这个指向数组的指针 + 行数 就能遍历整个二维数组 ====== int(\*pArray)[N] 没有M 那么 **就可以说二维数组的行可以在形参中省略**----- ***这样 即使没有行数 拼接成一个一维数组的时候 因为N存在 还是能够分清到底多大区域是一行元素 只是不知道有几行罢了***】



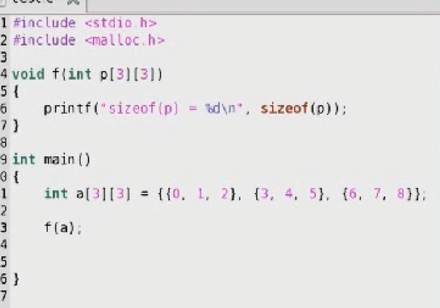


【所以 一维数组的退化规律 就是一维的[]去掉变成\* 放到数组名的左边

二维数组就是去掉离他最近的[] 然后 变成\* 放到数组名的左边】



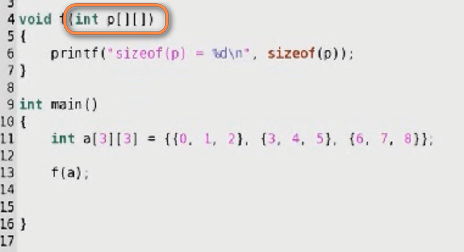
举例

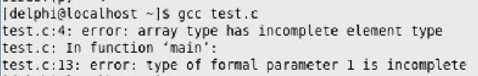


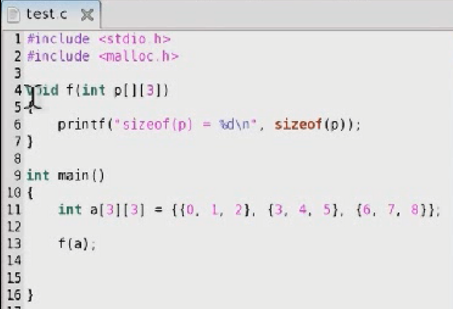
因为f(int p[3][3]) ---- 形参会退化成指针 所以 sizeof(p)就是计算指针占用的字节数 所以 值为4



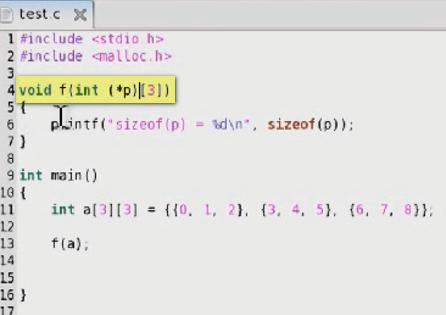
刚才说了 可以把最近邻数组名的维度去掉 那么 可不可以把剩下的维度挖掉呢？



看看编译结果：

 挖掉第一维可以 第二维不可以去掉

为什么？现在把这个形参退化到真实的指针形参



这个指针指向的数组 是什么样子？数组的大小是3，每个元素的类型是int

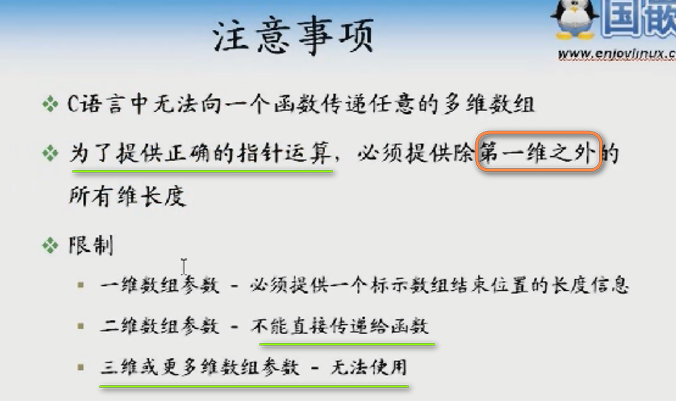
讲过数组的类型两个要素决定：数组的大小 + 数组元素的类型

===== 如果挖掉这个3

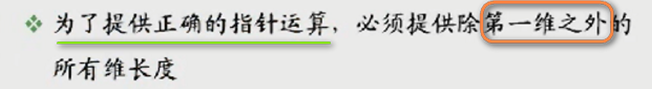
 看起来p指向了一个数组类型 但是没有大小 ---- -是指向数组么？---- 所以 编译不过

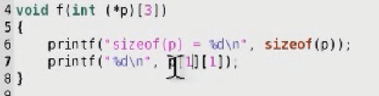
----- 所以 只能挖掉第一维的大小

**这样高维数组只有第一维可以去掉 ---- 因为数组类型需要大小**

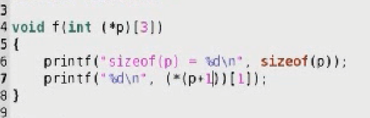


**解释下面这句话**





**直接变换成指针**



---- 可以看到 变换成指针之后 需要计算p+1 ---- p+1 === (unsigned int)p + 3\*size(\*p) ==== **如果没有 int(\*p)[3] 的3 和 int 那么 p+1就没有办法进行具体的计算**

P+1 ---- 就可以计算出向前移动12个字节

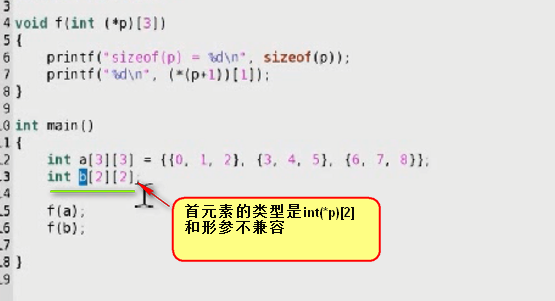
【可以看到计算公式的重要性】

编译器不让把第二维的3挖掉的原因 ----- 第一维为什么可以去掉？---- 第一维可以从外部传入

解释这句话：



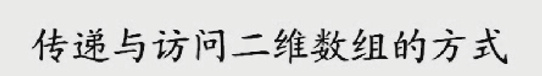
现在定义一个新的二维数组

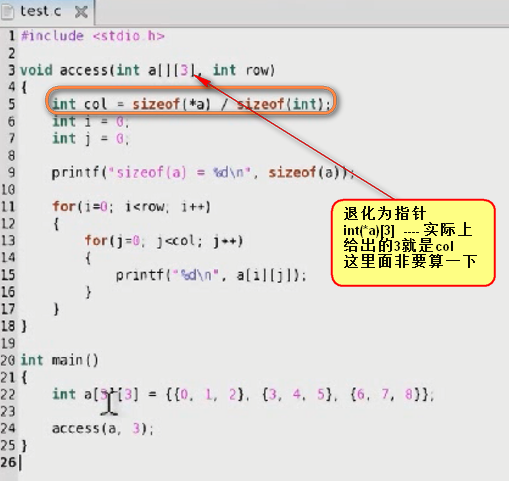


**此时编译器给出警告！**



在C语言中看到3维数组的机会很少





【想想 每一行的列数是一样的 ------ 所以 我要知道这行的总长度 然后除以每一个元素的长度 不就的出来这个列数么？

因为每一行的维度都一样 所以 我拿出来二维数组第一行的元素 也就是二维数组首元素 ---- 就是a[0] ---- 长度是sizeof(a[0])=3\*4=12B ***---- 这个一定是根据int(\*a)[3]中的3知道有三个元素***

每一个元素长度是sizeof(int)=4B

所以

Int col = sizeof(a[0])/sizeof(int) = 12/4=3所以有3列

现在想消除[0]这个计算的特殊化 所以 任何数+0 都是任何数 ---- a[0] = \*(a+0) = \*a

所以 就有 col= sizeof(\*a)/sizeof(int)

从sizeof(\*a)的物理意义出发 --- 二维数组名a也是二维数组元素首地址 ----- \*a就是二维数组元素 ----- 等价\*(a+0) ---- 等价a[0] ---- **可以看到\*和[]的互化非常重要** 】

【这个是如何动态计算二维数组有多少列】

***为什么申请一个动态二维数组 没有采用数组指针方式？ 而是采用二级指针+指针的方式么？【还是上节课的思考题】***