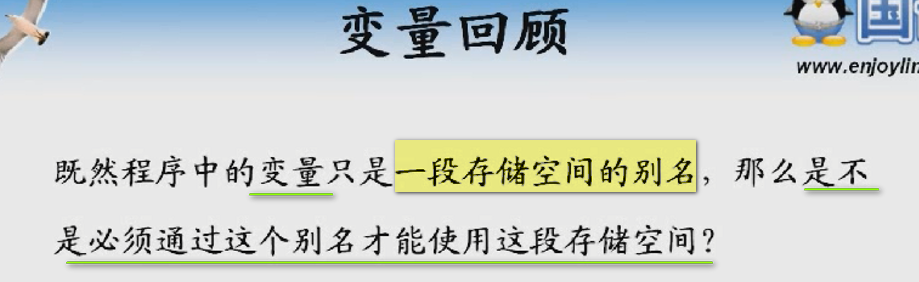
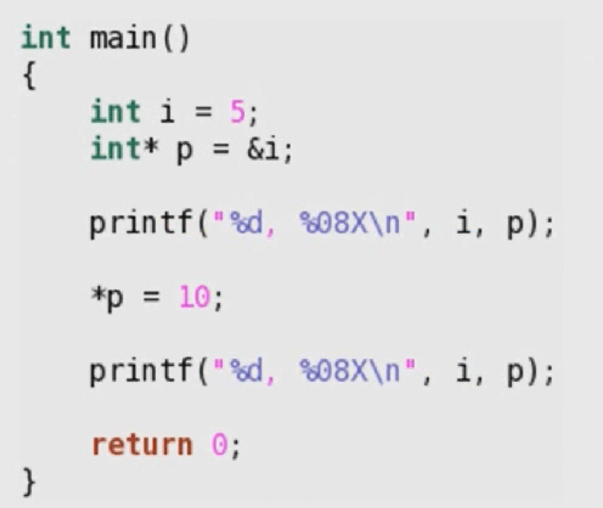


指针和数组是C语言中的最大的难点



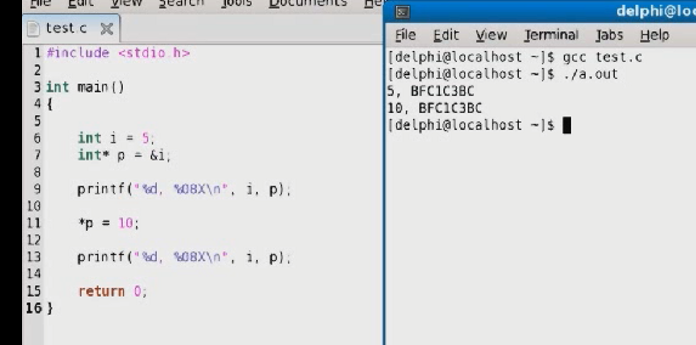


是否使用这个别名才能使用这段存储空间？



**即便变量名是一段存储空间的别名 要访问这段存储空间 或者修改这段存储空间的内容 未必要使用这段存储空间的别名 ----- 通过指针可以做到这一点**

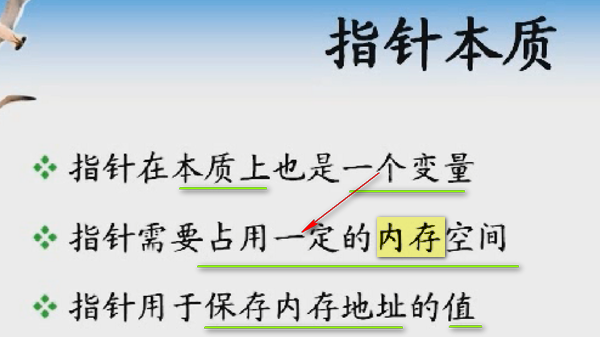
在linux里面复制运行



指针究竟是什么？

如果把int\*看成新的类型 这样 指针和普通变量的声明方式是一样的 ---- 唯一的不同就是有一个\*

如果把int\*看成一个类型 那么我们的指针int\* p 就是一个变量

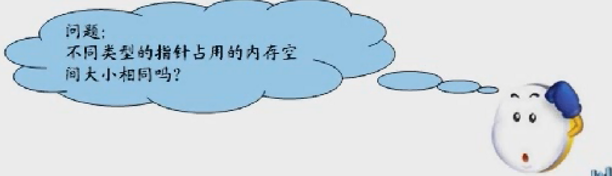


【指针变量简称指针 但是地址是常量的意思 ---- 所以 指针保存的是地址的值】

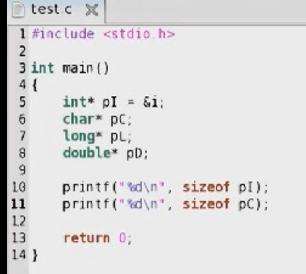
所以 指针是一种特殊的变量 ---- 我们的编译器认为这种变量保存的是内存的地址值 ---- 并且可以使用\*进行对应内存单元的访问

还有 变量有自己的类型 指针也有自己不同的类型

不同类型的指针占用的内存空间大小一样么？



很多人认为指针类型不同 占用的空间大小不一样 ---- 错

 结果 pI和pC的大小都是4 ---- 为什么？

**前提条件是 使用的电脑是32位的PC机 --- 对应的地址总线就会说32位 ----- C语言中 指针的意义是保存内存空间的地址值 ----- 指针变量中保存的值就是内存空间的地址 ---- 那么C语言就要保证 指针变量必须能保存任意一个内存的地址的值 ------ 所以 C语言才可以足够强大 作为底层的语言 写操作系统**

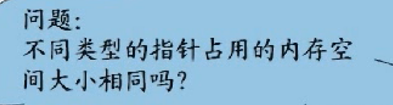
我们的指针变量占用的内存大小 不就是和地址总线一样么 就是32位 --- 也就是4个字节

这样 pI pC都占用4个字节

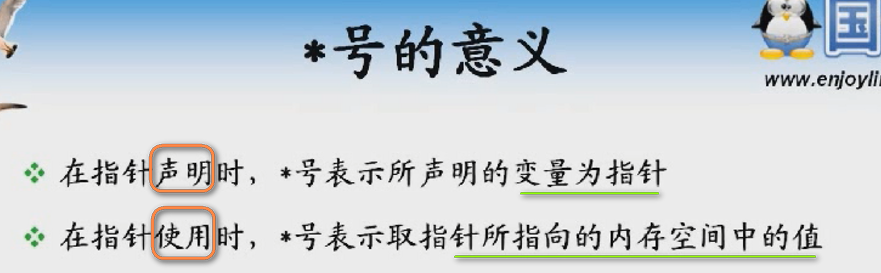
 

也都是4个字节

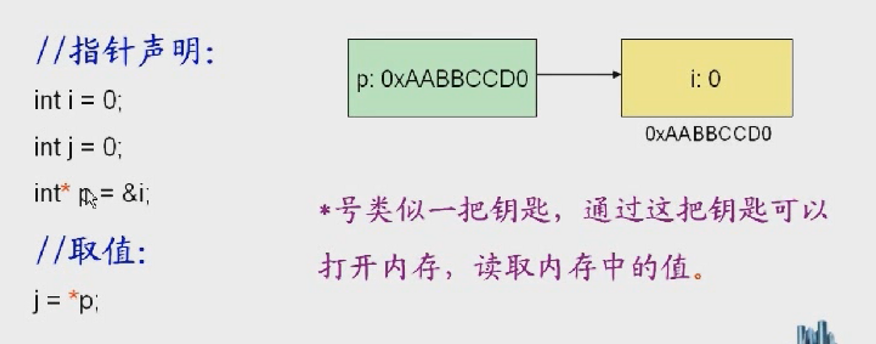
结论：目前32位PC机上面 指针变量总是占用4个字节

 肯定相同的

至于指针占用内存大小在每种机器上面一样么？不一定 原因就是指针要有足够能力保存每一个内存地址的值 在不同的机器上面 由于总线位数不一样 导致指针变量占用的内存大小不一样



\*还有一个用法 就是做乘法 ---- 这个是C语言中\*的三种用法

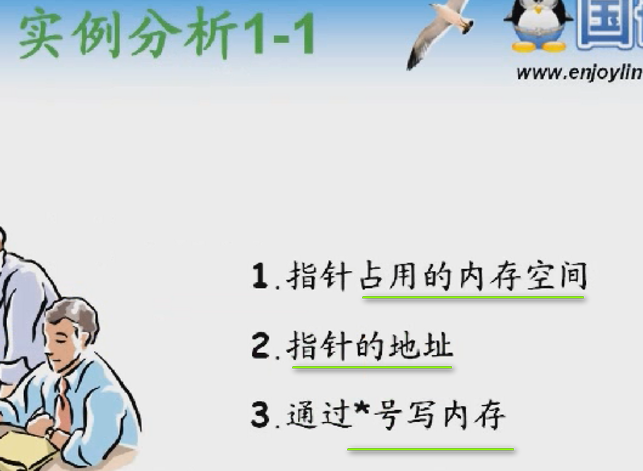


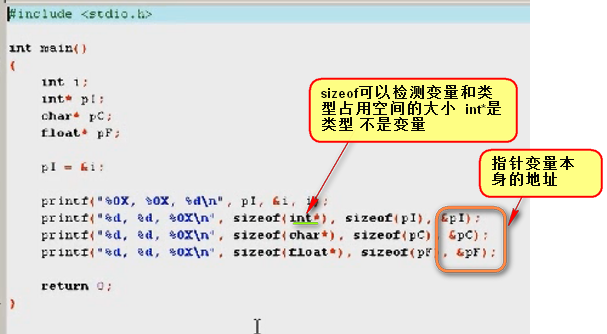
我们的p指向了 0xAABBCCD0 的内存单元 值就是0

这样\*就像一把钥匙 指针就像一个防盗门 ---- 通过防盗门到后面取值 需要钥匙打开 这样就过来取值

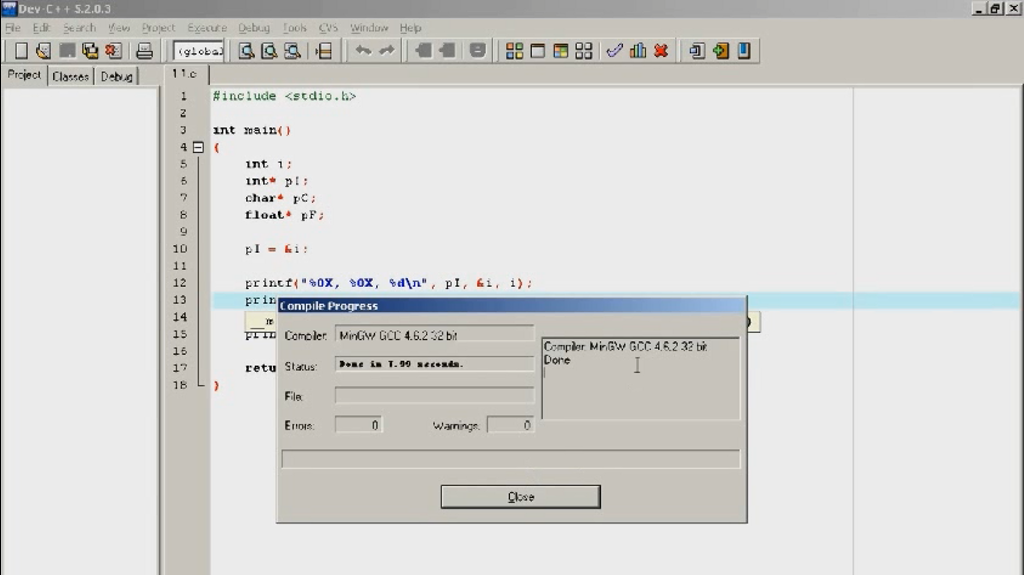
我们不同类型的指针 大小都是一样的 因为防盗门的大小一样 但是 防盗门后面的空间的大小不一定 ----- 地址总线 机器的位数

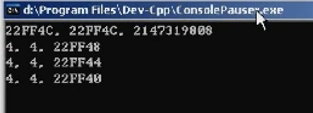
指针就是一个防盗门 防盗门后面保存的内容不一样 如何打开防盗门 这把钥匙就是\*

 这个我们在windows下面做 看看如何 ---- 指针也是变量 是变量就有地址



使用的编译器还是gcc

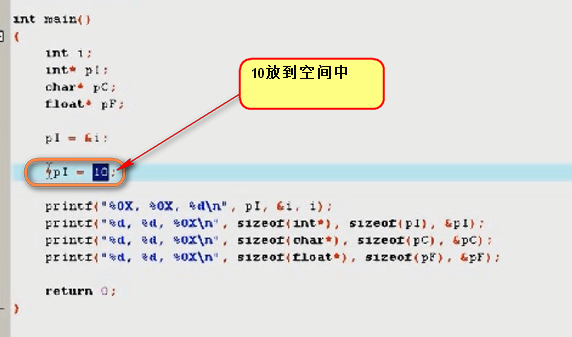


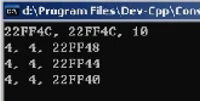
 --- 因为i栈上面 没有初始化 所以是随机值

sizeof(int\*)在windows也是4个字节 这是因为机器的原因【底层 C不是高级语言 中级语言】

指针变量也有自己的地址

修改这个i的变量的值





我们i的地址好像总是22FF4C ----为什么总是这个？windows os发现你总运行这个程序 ---- 所以 windows优化 就把这块内存空间分配给这个程序 ---linux未必这样

直接写地址的内存

现在发现22FF4C总是分配给这个变量i 直接写内存

 默认是一个整形 不是指针 --- 强制类型转换达到

 这个就是指针 ---- 要使用这个防盗门后面的内存空间 使用\*

 对这个进行赋值



运行看看i是不是100



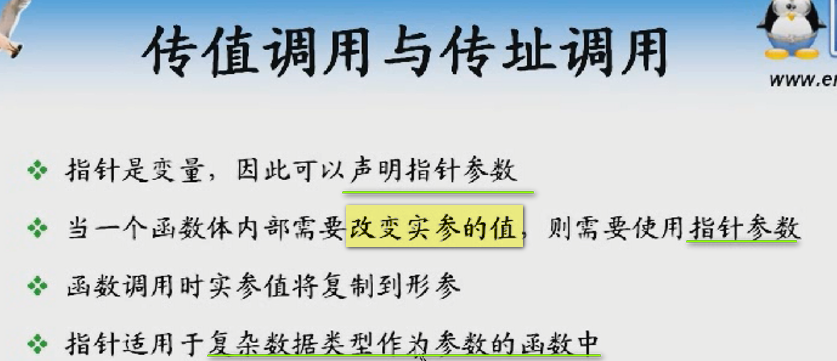
我们没有通过指针或者i本身赋值 ---- 但是利用了windows这个特性 ----- 强制类型转换成指针 这个地址在windows恰巧是i的地址

是不是必须通过变量名才能访问内存空间？

\*\*指针可以做这件事情

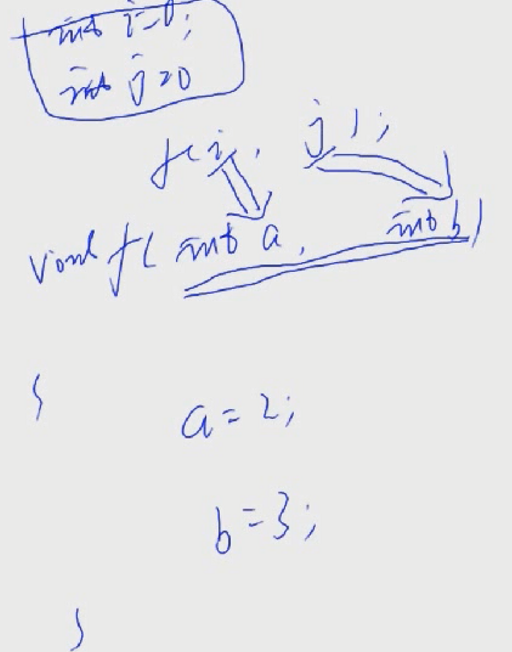
\*\*这片空间的起始地址直接写值

这在C语言中非常灵活



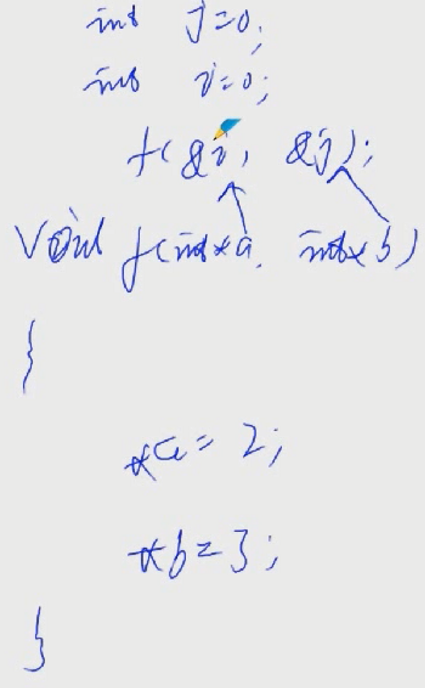


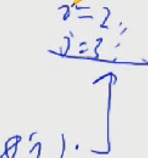
传值调用 和 传址调用的区别是什么？

 做了一个copy的过程 实参I copy到形参I j也同理 ----- 这样实参i j不会受到影响

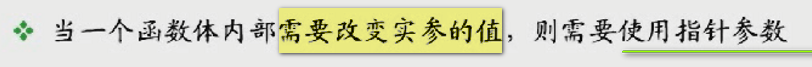
所以 函数体修改了i j与外部没有关系

现在用指针作为参数



调用完成后  传址调用可以改变函数外面的值

何时使用指针作为参数？



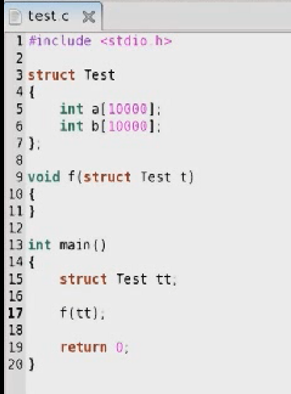
**函数体内部修改外部变量的时候 --- 可以使用传址方式** --- 一般的传值调用只是讲实参的值复制到形参 然后实参和形参就没关系了

**如果遇到复杂的类型的时候 出于效率的考虑 可以使用指针**

看一个极端的例子

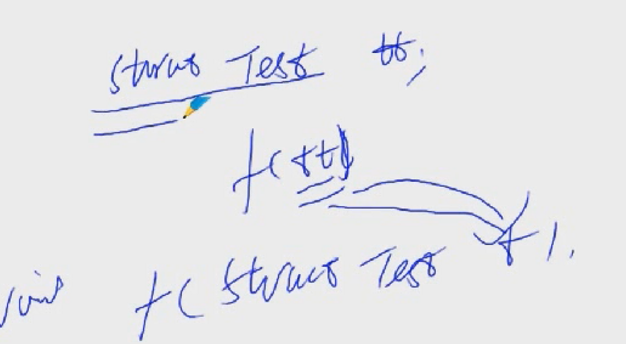
 非常庞大的结构体

有一个函数处理这种结构体

 进行编译 运行 也没问题

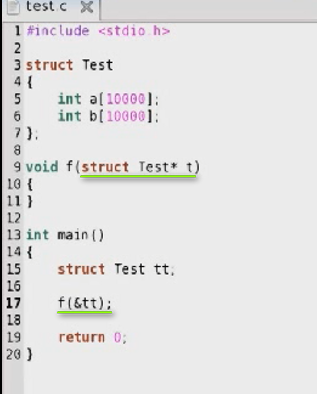
但是 对程序而言 我们传递的值 调用的f

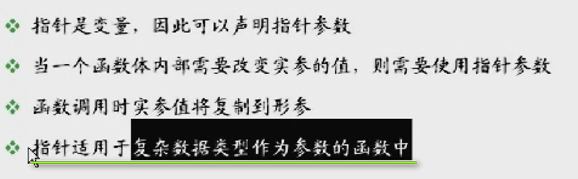
**理论上 这样会有一个copy的过程**



这个struct有2w\*4=8w个字节 ------ 这样copy就非常耗时

修改如下

 传递一个指针的地址值 4B 但是 传递值的话8wB 效率相差两万倍 -------

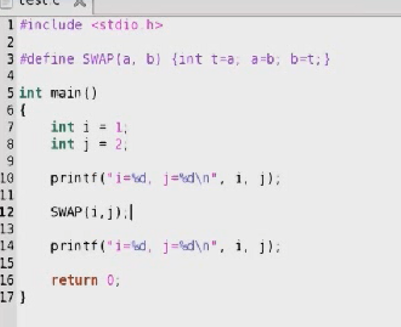


---------------------------



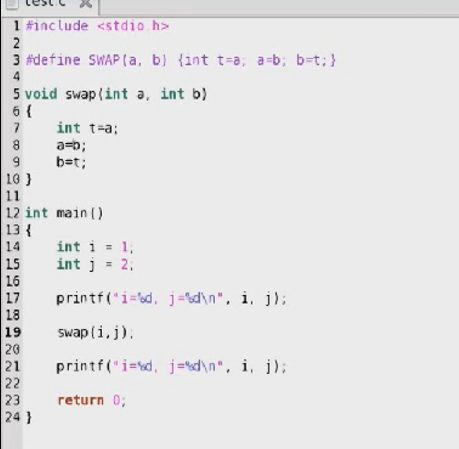
这个问题考过之前的学生 swap ----

先看如何使用宏实现

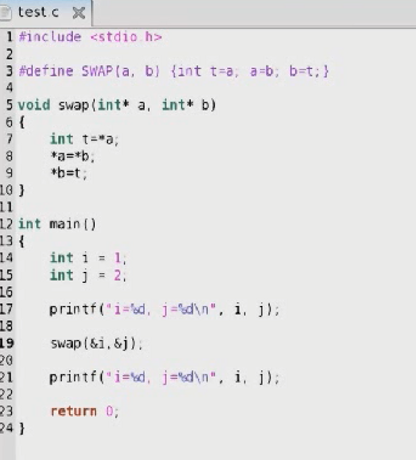
  宏交换了

**现在用函数进行改写**

他就直接把宏的实现 直接拿掉

  但是函数调用却没有交换 ---- 交换的是参数a b的值 没有修改外部的值

传址调用

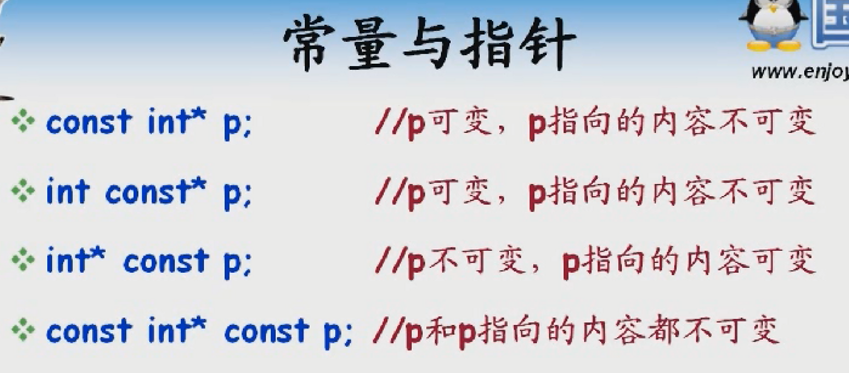
 

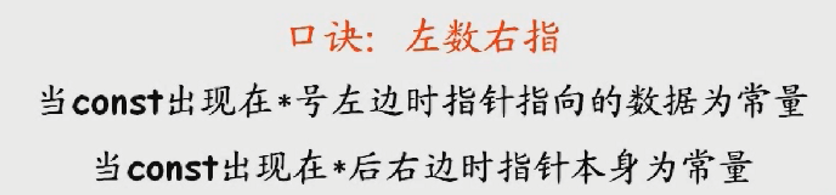
为什么宏可以？

swap直接替换展开 等于把这个语句 写到main中 这样 编译之前 就没有函数调用了

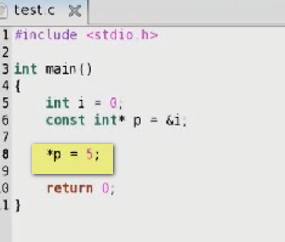
**swap不能进行传值 如果传值 交换的是内部的变量 外部的变量没有任何的影响 ---- 做成传址之后 就可以了**

**当需要函数体内修改外部变量的时候 使用指针传值 ---- 同时还高效**

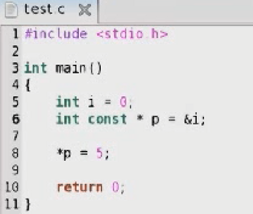
 面试 笔试经常考察



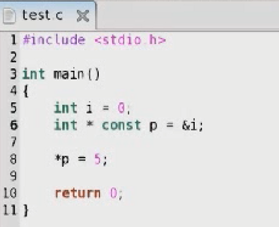
-------- 看看C语言编译器是不是用的这个规则



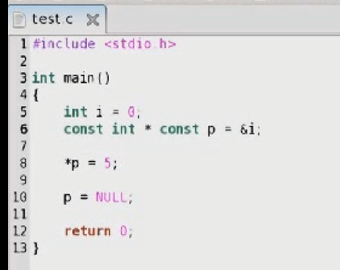


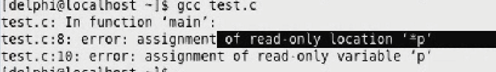
 编译错误一模一样 没有任何影响

左数这条规则是对的

 ----- 指向的内存空间可以改变 但是 指针不能改变

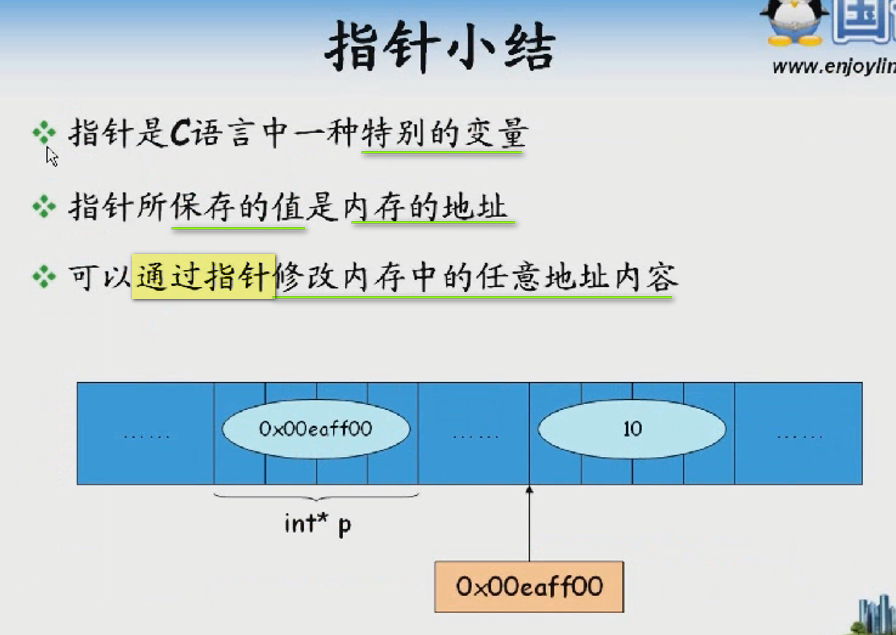
 编译不过





类似这种面试题 就是左数右指 看看左边 右边是不是有const

========== 小结



这个地方是指针的本质