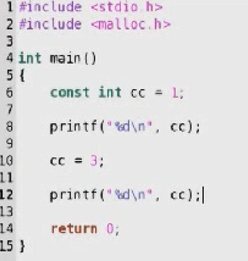
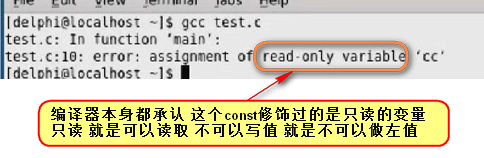


**既然只对编译器有用 对运行起来是无用的 ----- 那么也就是 程序运行起来之后 还是可以使用指针来改变const修饰的常量的值**

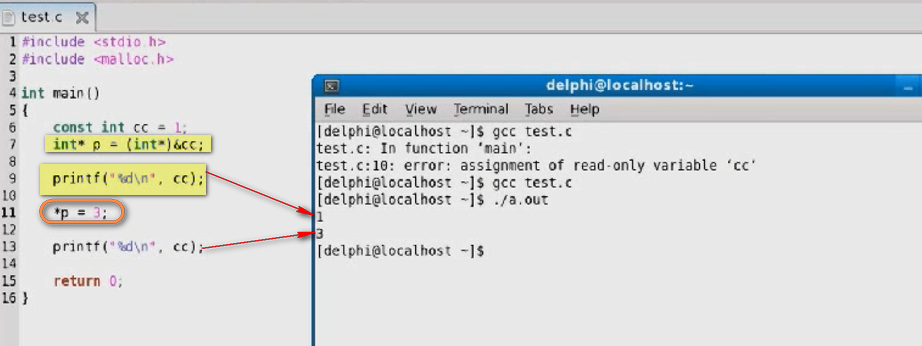
现在印证一下

 现在进行编译



程序的第10行 ---- 也就是给一个只读变量赋值

下面通过指针操作 给这个只读变量复制 read-only variable



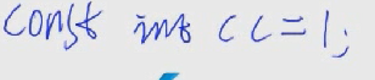
其实 我们的const并不是真的常量 而是告诉**编译器不**能作为左值 – 不能够给一个const修饰的变量赋值 --- 但是const修饰的还是一个变量 所以 一定在内存中占据空间 ---- 有内存空间 就有指针 ----- 有指针 就可以改变内存地址单元中的值

===== 这个就是C语言中const的本质

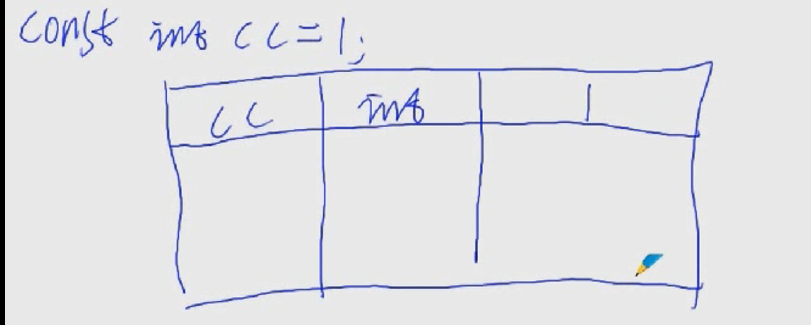
现在C编译器 对const做了一些优化 刚才是标准C中对const的处理

现在C编译器 或者C++编译器如何处理const常量呢？

现代的编译器里面 都会产生一张符号表 ----- 存放程序里面的变量名



C编译器中 看到这样的符号表 就是这样的一个数据项



下次会通篇检查有没有用到cc

第一个就是左值 ----- 如果出现在赋值符号的左边 ---- const 这个时候就会报错

第二个就是右值 -----

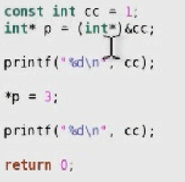
\*\*直接访问形式 ------- int cb = cc; ----- 这时候 直接从符号表中获取1进行替换 ---- 编译器就可以确认这个值为1

\*\*取地址符形式访问 ---- int\* p = (int \*)&cc; ----- 如果我们的程序中 有一个地方出现了取地址符 ---- 那么编译器就不会做这种替换 ----- 那么 就是运行时候到内存里面去取值 ---- 取值之后再给p

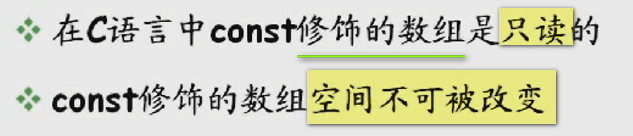
这个是现代C编译器对const的处理方式

但是C++编译器还有更复杂的处理方式

回到我们的程序中

 使用cc的方式有& ---- 那么后面遇到cc的地方 都是真真正正到内存中去获取cc的值 并打印出来

【C++编译器 在现代C编译器的基础上多做了一步 就是 除了兼容C的只读变量的内存地址空间以外 就是 出现cc的地方 直接编译器符号表直接替换】



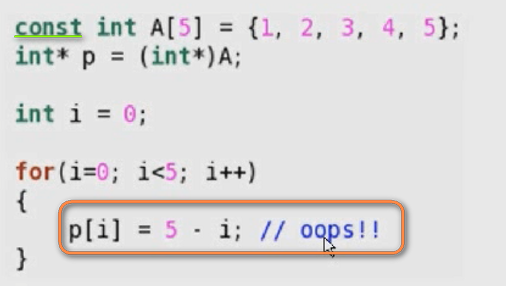
针对现在的C编译器不可以被改变 --- **如果改变 程序就会死掉**

但是在C标准中 我们的数组A在内存中还是分配了空间 --- **那么const是想说明我的数组中的每一个元素 是只读的 不能作为左值出现**

但是 通过指针还是可以改变数组中的值

----- 这个和const修饰普通变量不一样的地方就是 如果你这样做 那么 gcc或者g++这样的C编译器 就会让你的程序死掉 但是这个不是标准的C编译器规定的

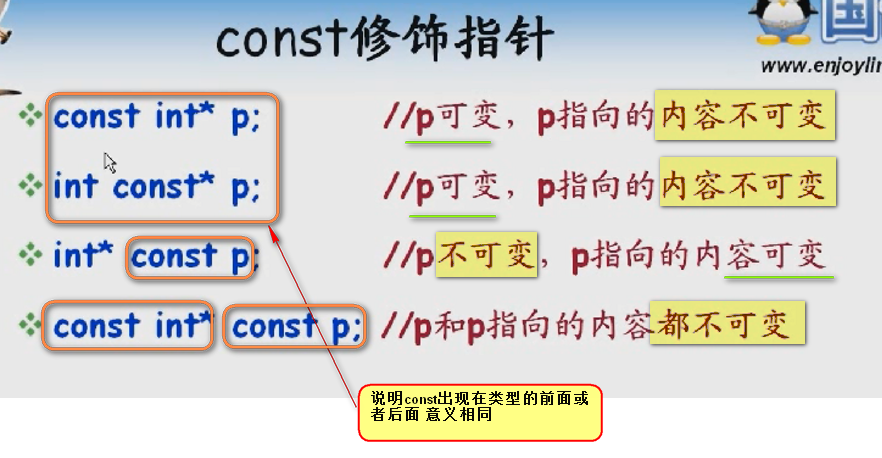
----

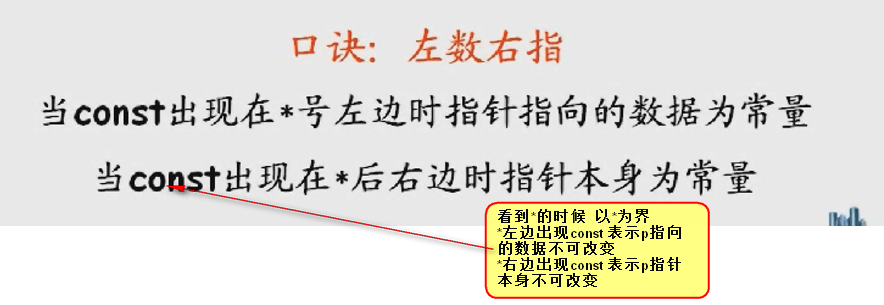


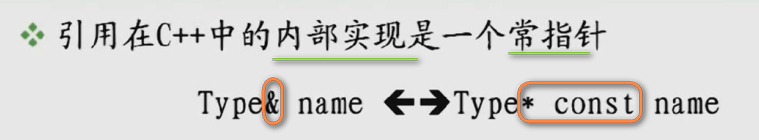
Const修饰的数组空间的值不可以被改变的 --- 虽然不是C标准规定的

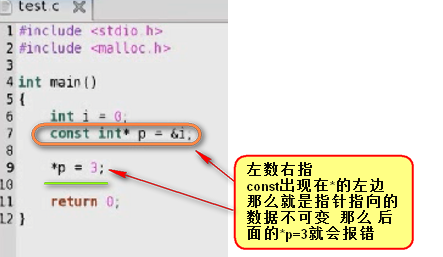
Const比较复杂的方面 --- const修饰指针

第一种 就是const出现在\*的左边

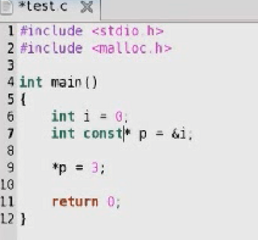




【 ---- 后面说的C++中的引用 就是左数右指的类型 --- 也就是指针只能指向这一个固定的内存单元 ----- 和java一样 final的变量必须声明的时候 就要完成初始化 所以 讲解引用的时候 说 引用必须要其他变量给赋值才可以 和java的是一个意思】

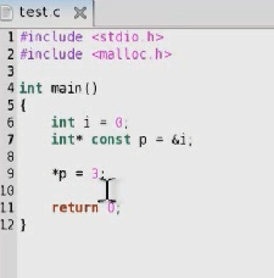




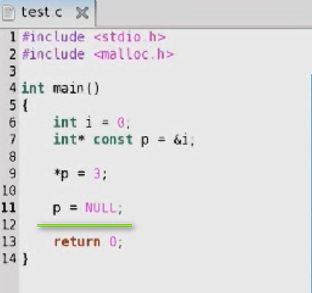
 还是出错



所以int const \* 和const int\*一样

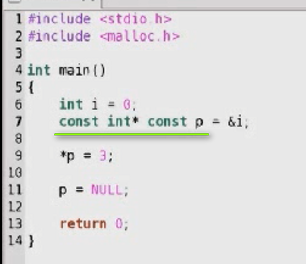
 编译通过 ----- 左数右指 int\* const 说明 指针本身不能被改变 但是 指针指向的内存单元的值可以改变 所以 \*p=3就是正确的

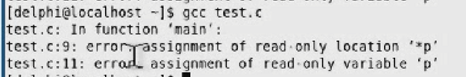
现在修改指针本身 赋值为NULL





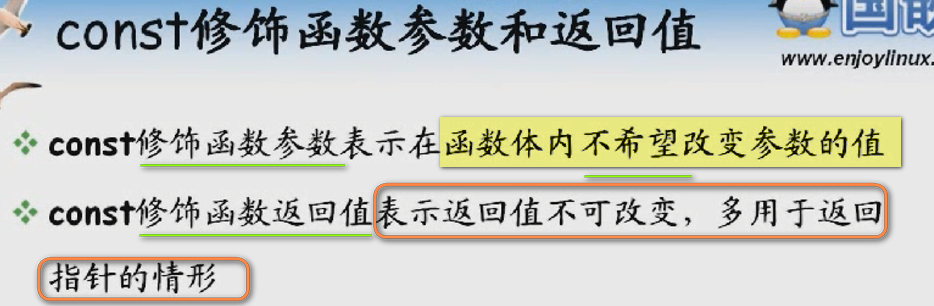
添加一个组合



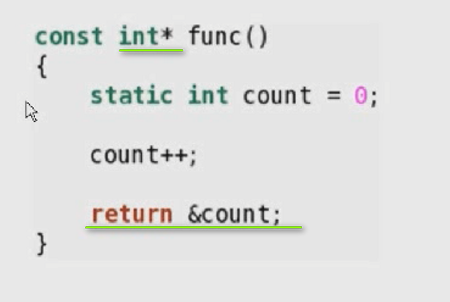


【\*p表示指针指向的内存单元】

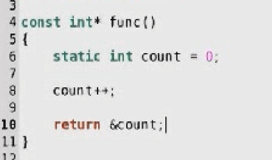
----- const修饰函数的参数 以及 const修饰函数的返回值



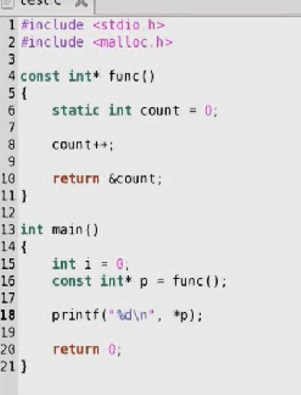
如果有一个函数 返回一个指针 指向的内容不允许被改变 ---- 如果一个函数返回了一个const类型的指针 表示这个函数返回的内容不允许被改变



【如果是引用的话 就可以直接return count; 就可以了 如果这个count是局部非静态变量 那么这种是没有意义的 所以 引用用起来比指针易读】

 返回的是局部变量的地址 但是 count 因为static修饰 所以不是在栈上面分配空间 而是在全局数据区分配的空间

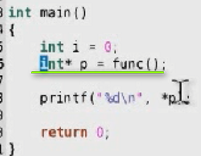
所以返回static的局部变量是合法的

 编译通过



很正常

如果我们接收的时候 使用int\* 而不是const int\* 会怎么样？

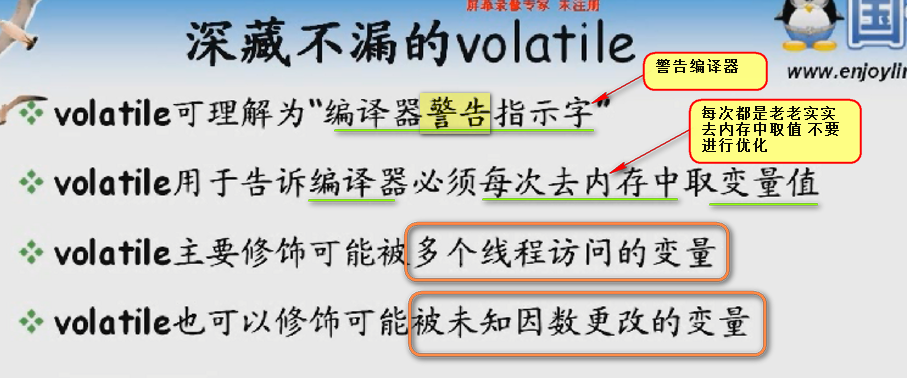
 

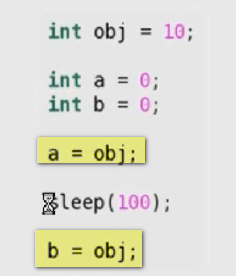
说指针返回的类型是不一样的

虽然只是一个警告 但是 对于程序员 任意一个警告都要当成错误来进行对待

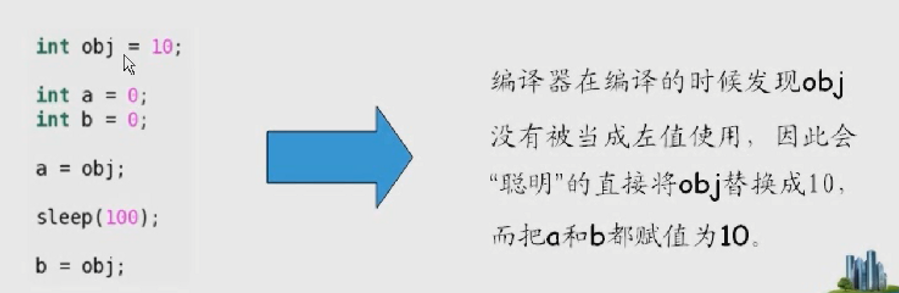
当我们的函数的返回值有const修饰的时候 要搞清楚 我的这个方法是想返回的什么内容不变 --- 是数据不变 还是指针本身不变

========= volatile （Java中是高级应用）



 就是 obj自从初始化之后 都是当做右值 -- 意思就是都是出现在程序赋值符号的右边

这样编译器就做一个优化 默认obj的值不会被改变

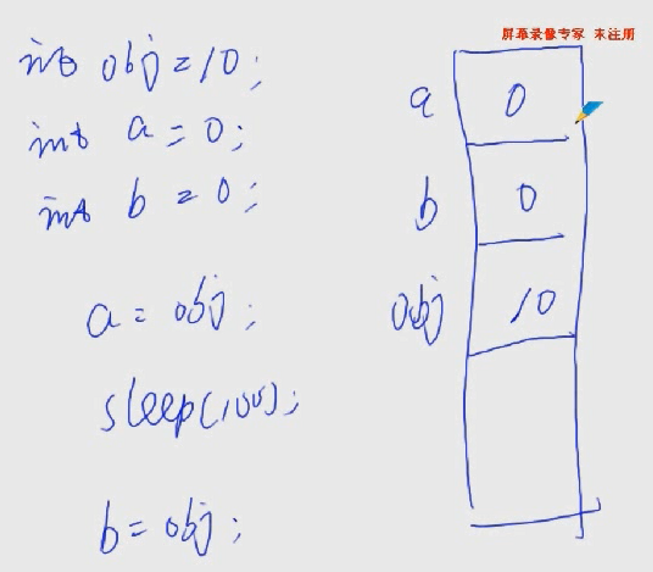


如果程序就这么几行代码 都是没有问题

但是 如果出现这样一种情况

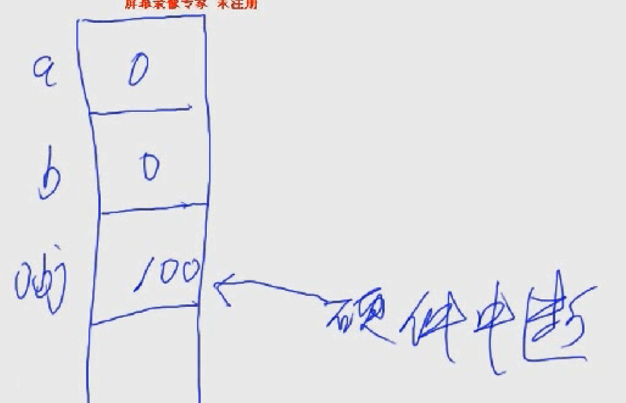
在我们的程序里面

此时 obj a b在内存中的值画出来

 当没有声明为volatile的时候 我们的编译器 就会自作聪明 把a b的值都替换成10了（还没运行呢 值就被替换了）

因为Obj从来都没有作为左值 这样使得程序运行过快

但是 如果在100ms之内 产生了硬件中断 然后中断处理程序改变了这块内存的值、



当100ms过了之后 之前 由于编译器做了优化 直接把10替换了obj ---- 这样b得到的值还是会是10

但是 我们等待就是为了硬件中断 就是修改obj的值 但是由于没有使用volatile 编译器就自作主张的优化 导致程序没有出现预期的结果

如何修改这个程序？

有一个值 被未知的因素 或者 多线程 修改的时候 需要使用volatile



问题：

