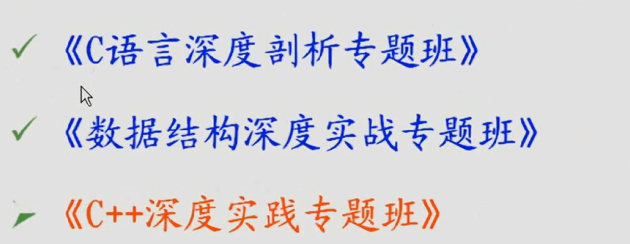
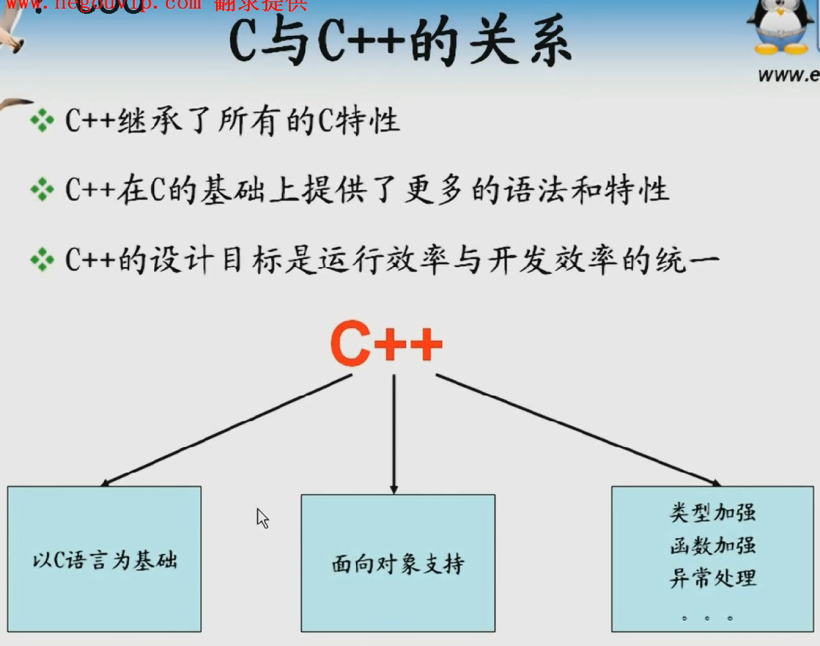
第1课 – c到c++的升级



C++继承了C所有的特性 提供了更多更好地语言特性 把C中不好的语言特性都删除了



C语言是专门开发操作系统出现的 所以 设计目标只是运行效率高效

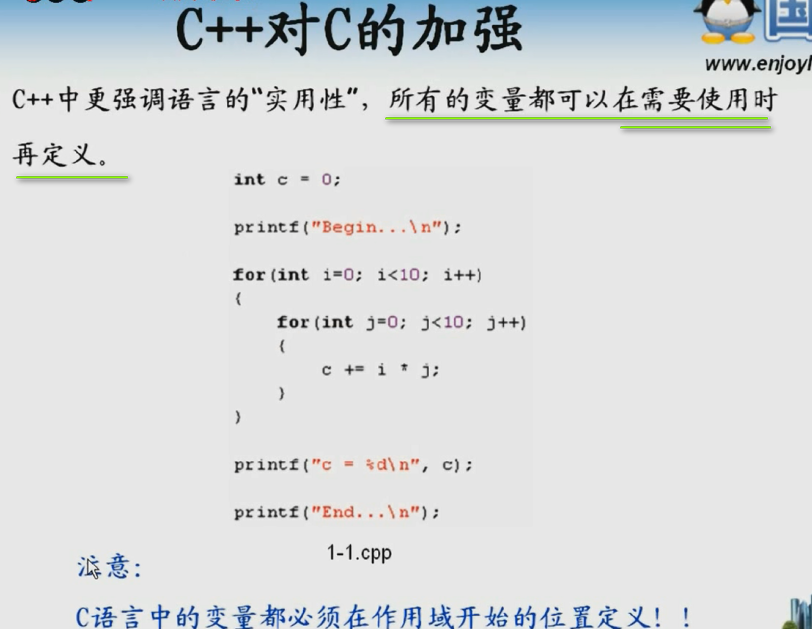
C++是运行效率和开发效率的统一 + 平衡

**---- 开发现代的软件系统 C++比C更合适**

第一个+表示C++从C语言而来 C++是C的继承者

第二个+表示C++**支持面向对象方法论的所有特性** 以面向对象方法来进行软件开发和软件设计 --- 代表C++对C语言进行了加强 类型加强 函数加强 异常处理

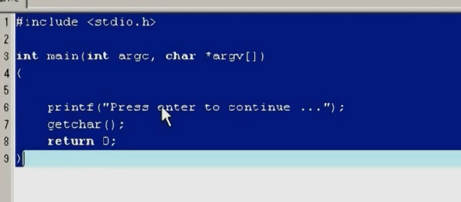
**\*\*1 --- C语言中 所有的变量都要求在作用域开始的位置定义**



之前的C语言好像没有这么写的

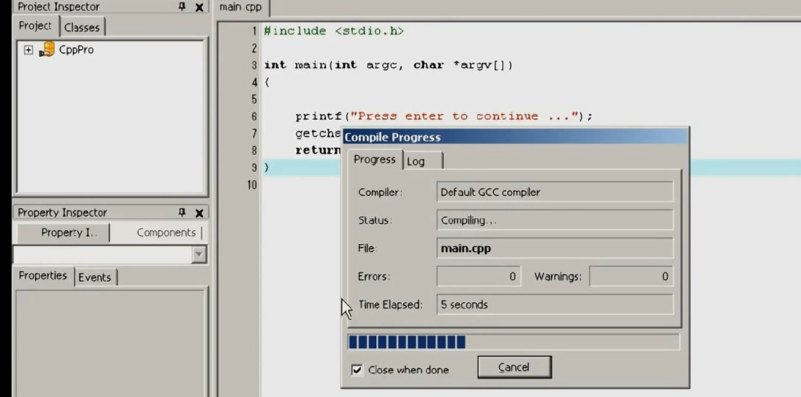
构建一个C++工程

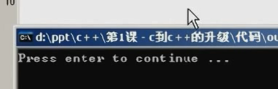
和一个C语言的工程



把这段C语言代码复制到C++工程中去

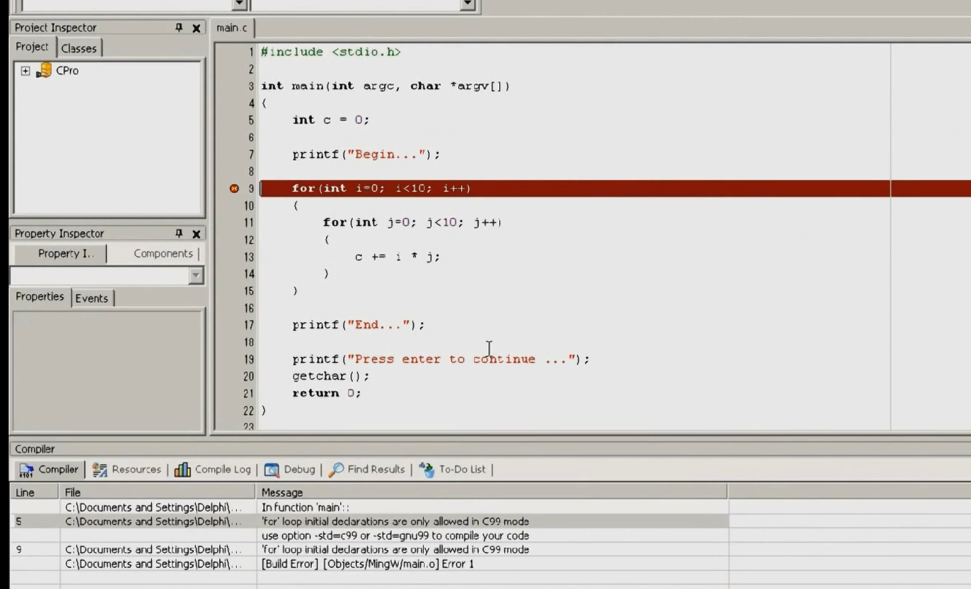
用C++工程编译标准的C代码





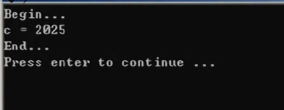
**这个例子 C++ 确实兼容了C语言 ---- C++工程编译标准的C代码**

回到PPT上面的例子 把这个保存在C工程里面

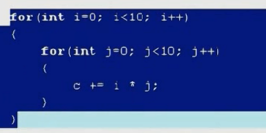


报错了 说了 在for循环中不能包含变量的定义

但是 到了C++工程中编译通过 运行出来了

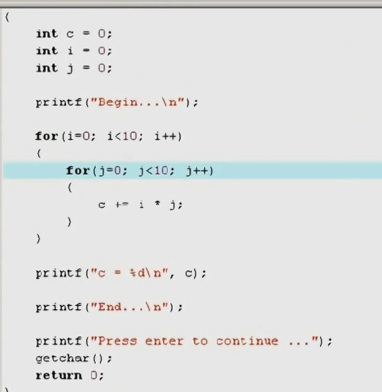


C++中 这个程序完全合法 但是C中 不可以



这么写 j的作用域就是for循环 i也是

整个程序中嵌套的i j 都放在for循环中 --- 其他地方没用到 ---- 所以 就不用传统的C 把所有的循环变量也写到程序的最开始

 ---- 这样写好麻烦

因为程序开始 的时候 无法知道会用到哪些变量

这样 可以提高开发效率 ---- 如果一个函数100多行 是不是要把所有的变量都放在最前面 这样是不合适的 应该用到什么变量 就定义什么变量

但是 C语言 就不能这样

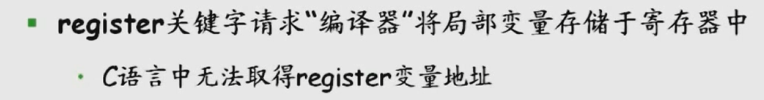
为了提高实用性 C++就做了加强 ----- 这种在真正的实际开发软件中 效率会提高

==============================================================

**\*\*2 --- Register关键字的变化**

C语言中 register关键字的意义就是**请求C编译器 将一个局部变量存储在寄存器** --- 为什么要有这个请求呢 老师认为这是一种优化 --- ***存取速度变快 整体效率提高***

C做了假定 如果register修饰局部变量 ---register关键字就会请求编译器讲局部变量存储到寄存器中



因此放到寄存器中的变量没有地址

C就直接规定 不能取得register变量的地址

C++中的register就是为了兼容C语言的程序

C++是现代编译器 --- 会自动主动做优化 --- **不会像C编译器那样 需要外部指定 C++编译做的优化 比我们自己请求C语言编译器要强大的多 ---- 主动 自动做优化**

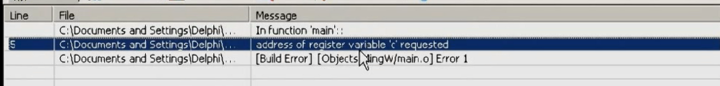
**--- 所以register关键字在C++中可以删除 可以不用了**

**为什么register在C++中没有被删除呢？ 仅仅为了兼容C语言**

因此C++中 编译器发现有register时候---- 同样可以把这个register修饰的局部变量当成内存变量 是可以获取地址的 --- 只要这么做 c++编译器就知道 不能放到寄存器中 --- 这样register修饰变得无效

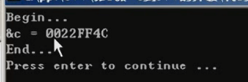


看看C语言编译器



这个C语言编译器直接说 对寄存器变量获取地址是错误的

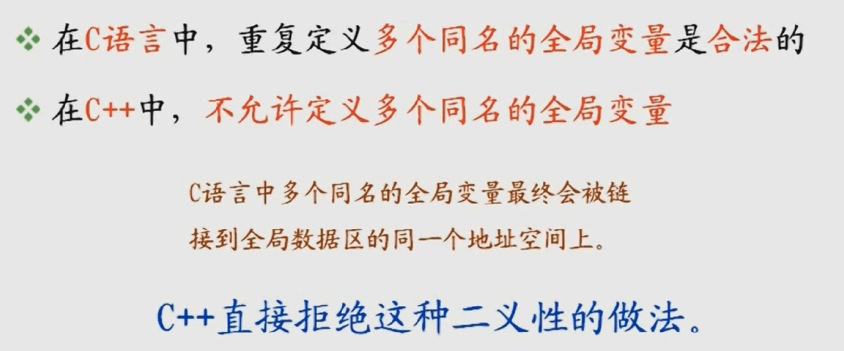
把这段代码放到C++工程中

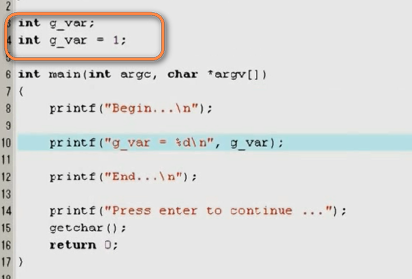
 编译通过 并且地址出来

这意味着register关键字在C++中变得鸡肋 要不要无所谓

 这样一段的普通的代码 C++编译器 也可能把这个C变量自动放到寄存器中进行优化

**\*\*3 ---C语言中 可以重复多个同名的全局变量是合法的 ---- 但是C++中不允许定义多个同名的全局变量**

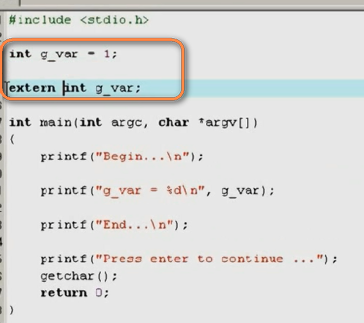


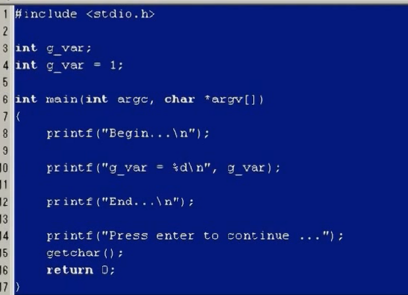
 C编程中 编译合法 ---- 这里面是在同一个文件定义两个同名的全局变量 --- 不同的文件里面同样可以定义同名的全局变量

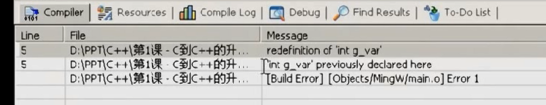
现在放到C++工程中

你知道多个文件中都定义了 究竟在哪个文件定义中才有效么？

最正确的写法 第一个文件里面这么定义 第二个文件中使用extern

 这个g\_var变量在其他文件中定义过

在C++编译



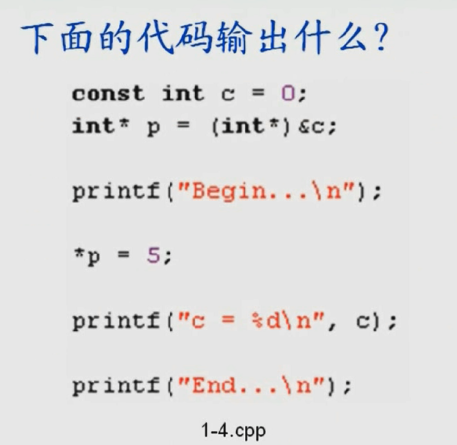
告诉你重定义了g\_var的变量 直接被拒绝

只能使用extern这种无二义性的方式



可读性也变强

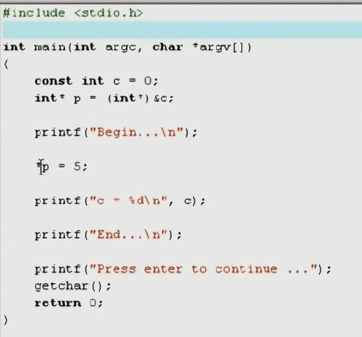
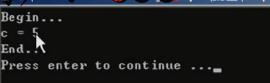
**C语言中 虽然可以定义多个同名的全局变量 但是最终会被链接到同一个地址空间 就是一个全局数据区 g\_var虽然定义了两次 但是只有一个存储空间**



这里面是声明一个常量 c --- 利用指针修改这个只读变量的存储空间

 学过之前的课 这里面打印的是5【？why】

C工程编译

 因为这个是只读变量 还是有存储空间的 --- 所以  真的变成了5

**---- 所以 C语言中const只是一个冒牌货 是可以被修改的**

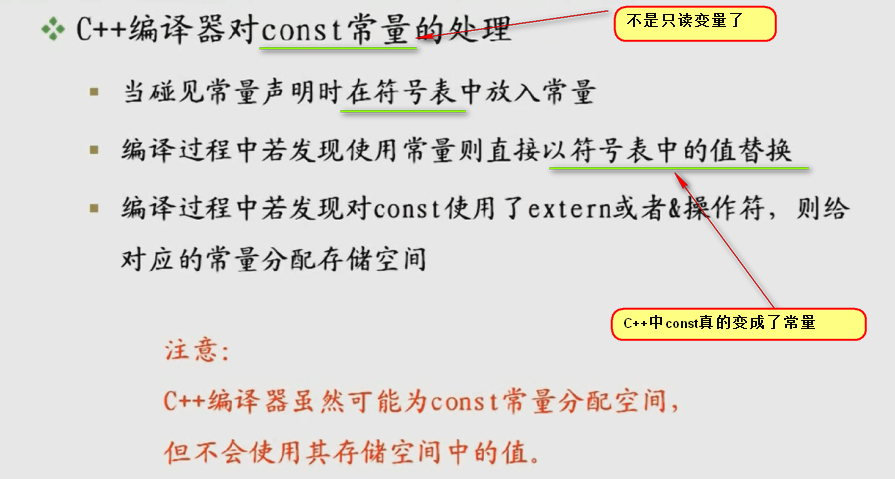
现在把这段代码放到C++工程中

编译是通过了 但是 看到 运行结果还是最初的0



并没有变成5

C++对于const做了一个很大的加强

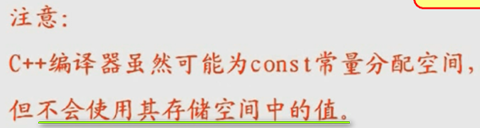


C语言中 const修饰的变量就叫只读变量 ---- 但是C++中 不能叫变量了 **真的叫const常量**

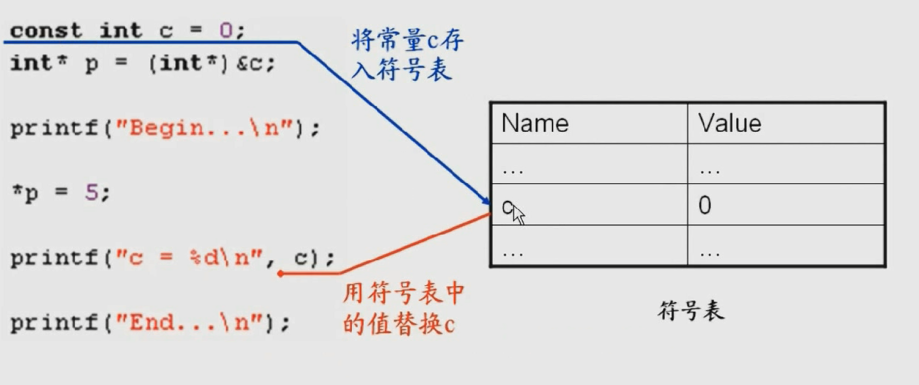
那么 C++中如何解释可以获取常量的地址呢？



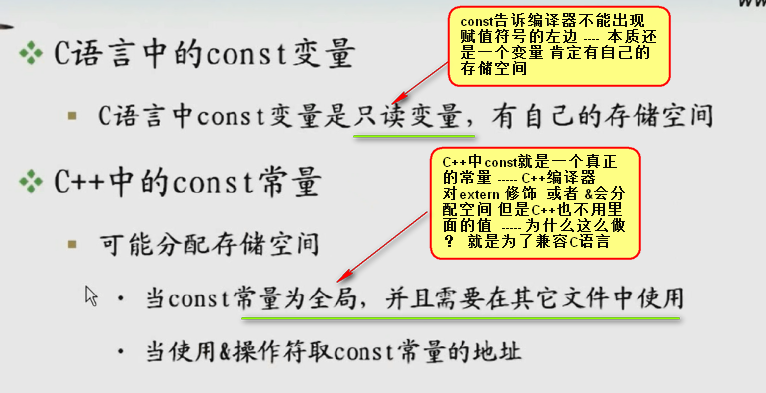
C++编译器一般不给常量存储空间 ---- 只有这个const使用了extern或者&的时候 才给这个常量存储空间 ---- 既然如此 为什么 这个地方修改了这个常量的值 结果没有体现出来呢？

原因就在于 C++编译器有可能为其分配空间 ---- 前面说了分配空间的情况 但是 分配了空间 又不会使用里面的值 

也就是遵循了下面的过程：

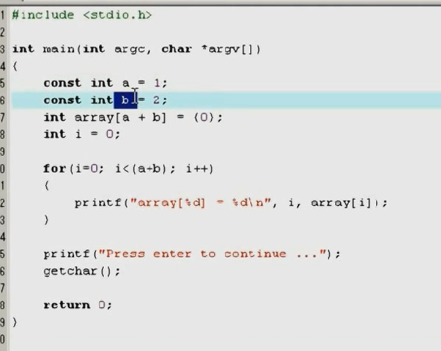


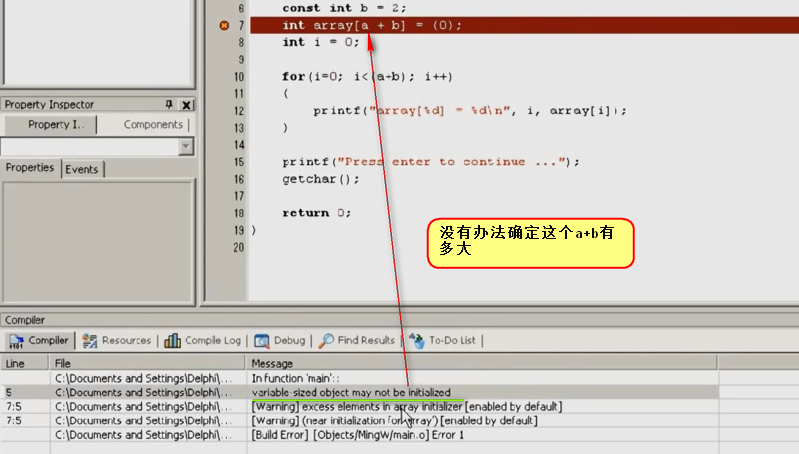
\*p指向了c的存储空间 但是并不会使用里面的值 虽然改变了 但是不会使用 这就是C++中对const的加强 ---- c++中const不是冒牌货 是真的只读常量





下面是C工程里面的

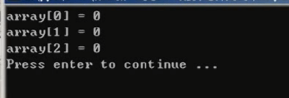




没法确定这个数组有多大 ---- 因为a和b在C语言是const的 但是c语言中的本质还是变量 那么c编译器还是无法知道这个变量的值有多大 ---因为变量 只有运行的时候才知道

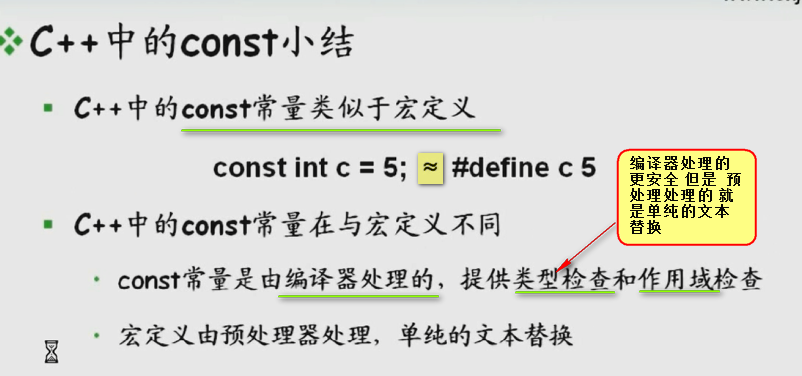
所以 编译器无法确定数组的大小 所以报错了

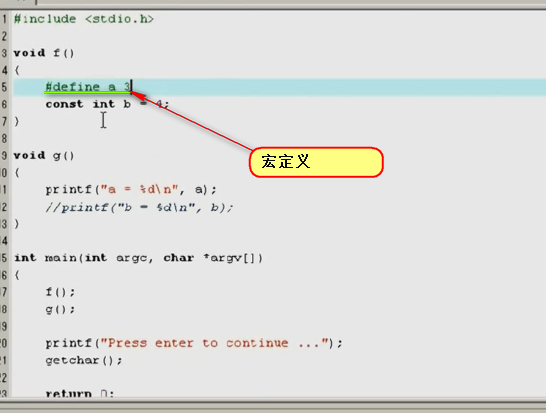
C++就编译通过了



发现a b是常量的时候 就把a b放到了常量表

这样 遇到a b的时候 就把对应的常量表中的值替换 ---- 这样就反应了c++中所修饰的 就是真真正正的常量 不是c语言中的只读变量





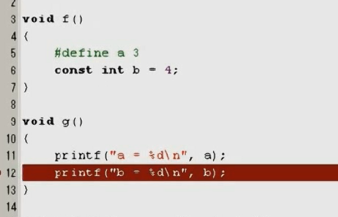
A的值是在函数f里面定义的 那么在g()中 按理说 不应该访问这个a的值

 但是 C++编译器是可以通过的

并且打印出3 

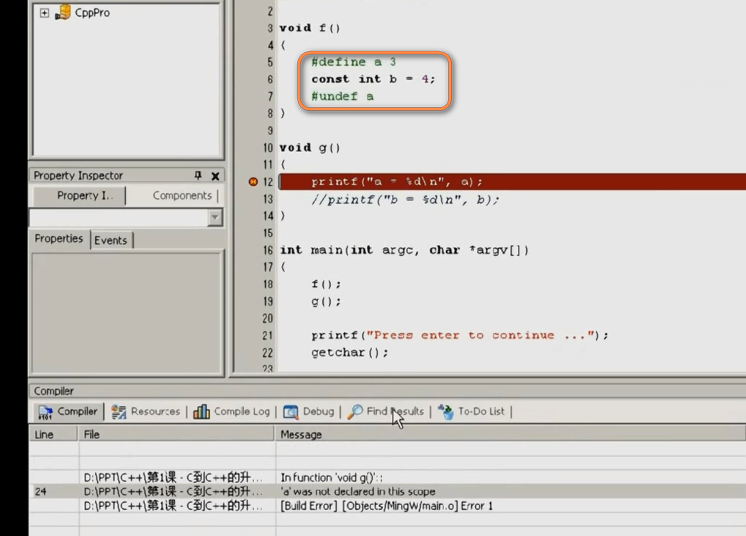
既然打印出来了 ---- 如果这样的代码 写代码的人 就希望a定义的这个3 就希望有局部的作用域 ---- 但是 不幸的是

预处理器就进行简单的替换 --- 那么编译器在编译之前 预处理器就进行简单的文本替换 所以 编译器处理的时候 这个g()中打印的语句 出现的a的地方已经被替换成了3

但是 b这个就不行了 因为是编译器的行为 编译器检查类型和作用域 ---- b的作用域不在g()里面 所以 就会报错



技巧：如果希望这个宏定义的常量a就在f()这个函数中 那么 就在函数的结束的地方 使用#undef 就可以了



这样很麻烦 还不如直接使用真正意义上的const常量呢

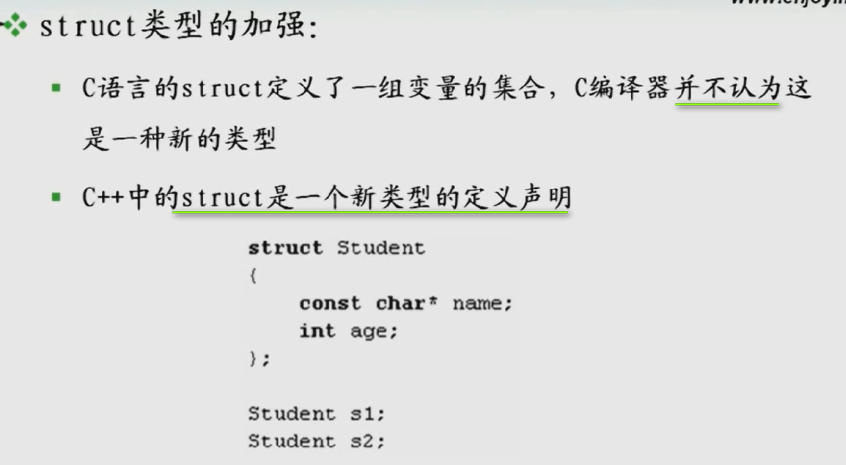
最重要的const的变化就没了

--------------------- C++的加强还有一个就是对类型的加强

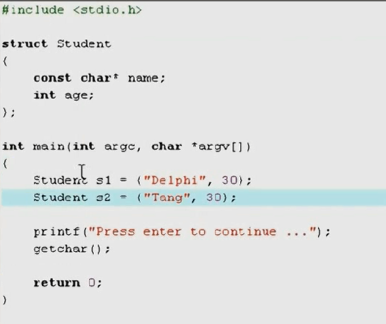
C语言中 使用struct定义自己的类型

C语言不是定义了一个类型 ---- C编译器不认为这个是新的类型 只是认为这个struct是新的变量的集合

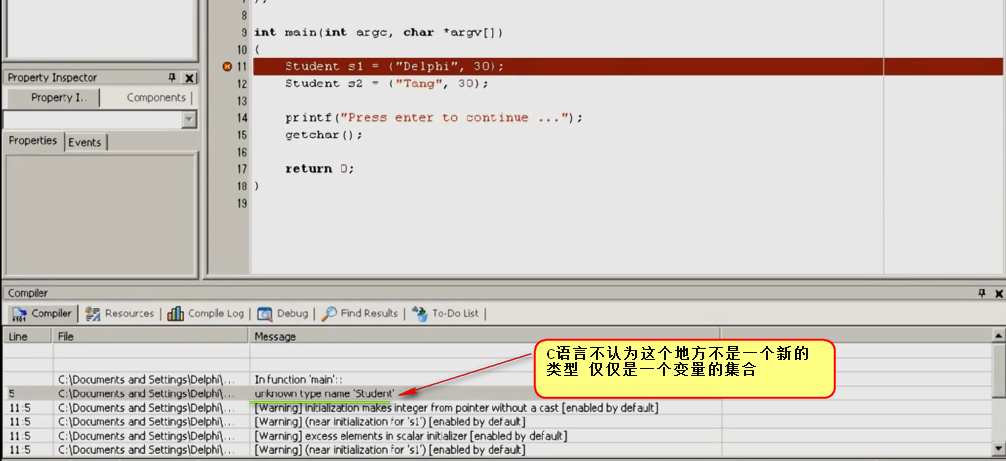
但是c++认为struct是定义了新的类型



这里面 C++认为Student是一个新的类型

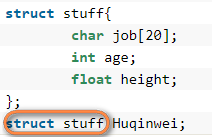
 这样 这个就在C++编译器中编译过去了

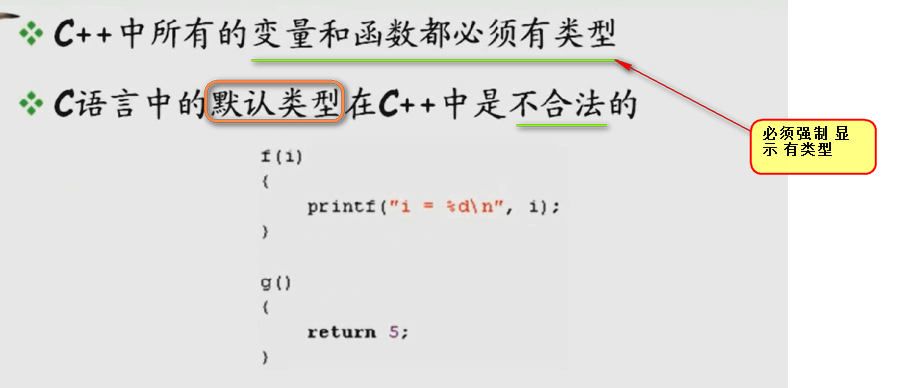
现在相同的代码放到C语言编译器中 进行编译



C++是一个强类型的语言 类型的检查比C更加严格

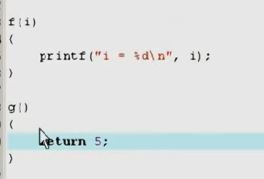
【在C语言中 定义完的结构体 要使用的时候 必须加上关键字struct一起使用才没不会报错 因为他不认为struct定义出来的是新的类型 所以 必须跟上struct这个关键字本身才知道这个Student是一个struct类型

 但是 在C++中 这种方式定义完成的话 可以直接使用stuff Huqinwei;】



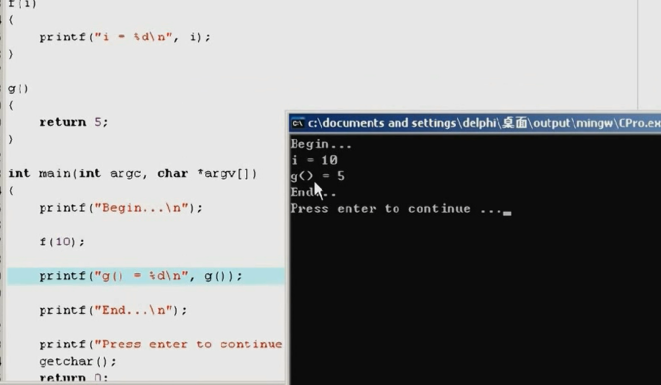
假设f(i)和g()合法 --- 那么 f()的返回类型是什么 g()可以接受几个参数？

在C编译器中 下面编译通过 合法



如果这两个函数合法 这个i是什么类型？

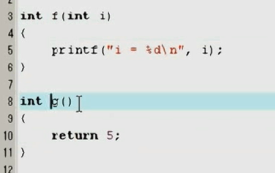
调用一下看看



意味着 这种写法早就应该被淘汰了

这种I 默认是int

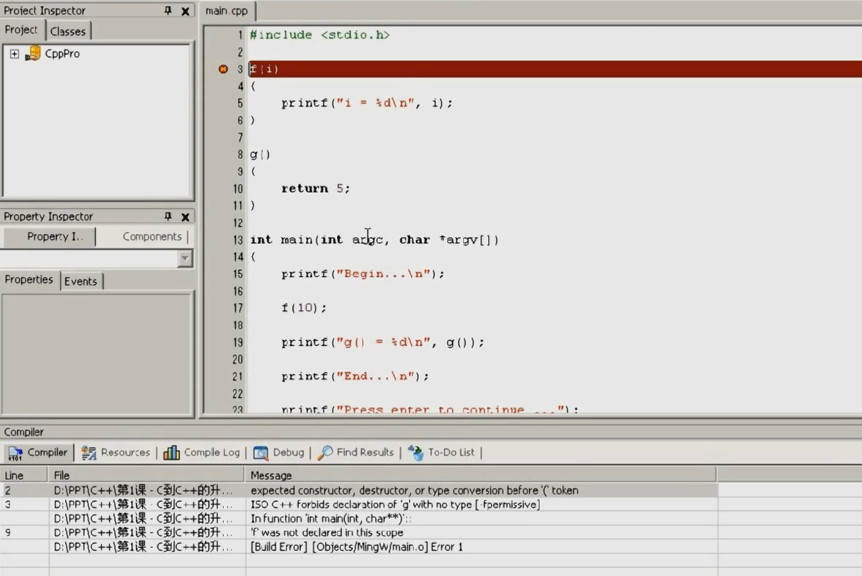
这两个函数等价如下



这样 g()这个时候 没有问题 但是 f()就有问题了 -----因为 f定义了有返回值 但是 实际上缺没有返回值【java中 这样就是不可以的】

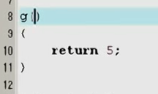
但是 都能通过的 ----- 竟然允许函数如此随意 太随便了

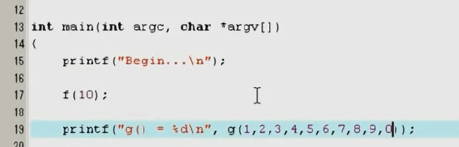
同样的代码 放到C++编译器中



C++告诉你 不知道f是什么

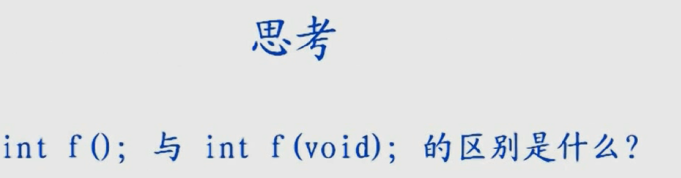
C++不支持C语言的垃圾特性 --- 够垃圾 C语言中这种非常不好理解

 这个没写 是不是不接受参数呢？

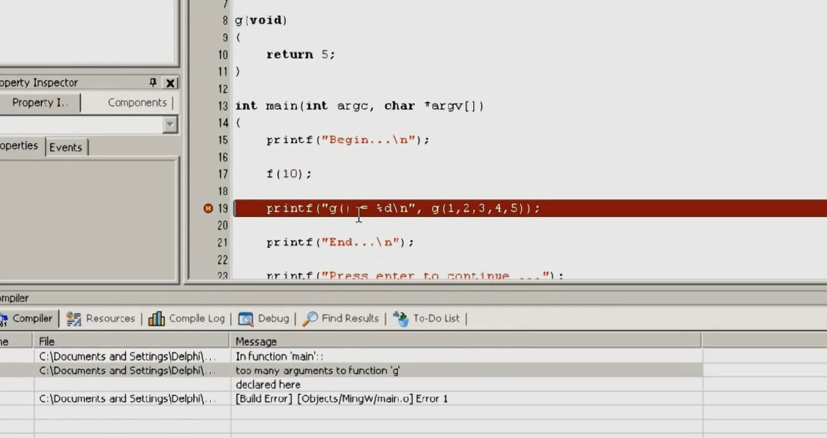


C语言编译器中 都能编译通过 这样的特性非常不好

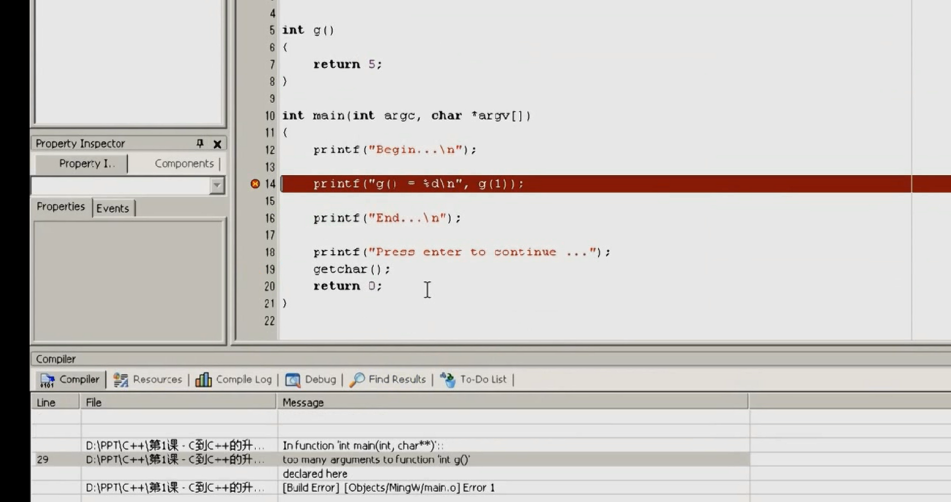
C++中 认为对C中的很多加强都是合理的



答案：f()中写上void代表C语言中的函数不接受任何参数

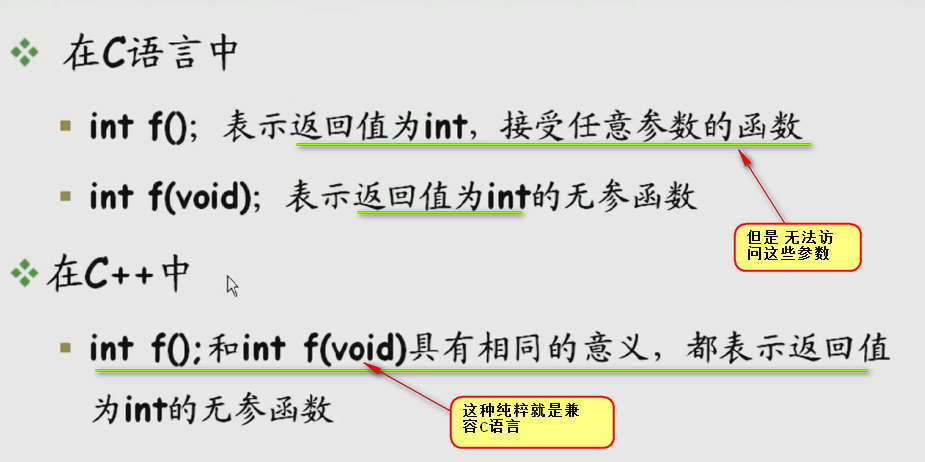


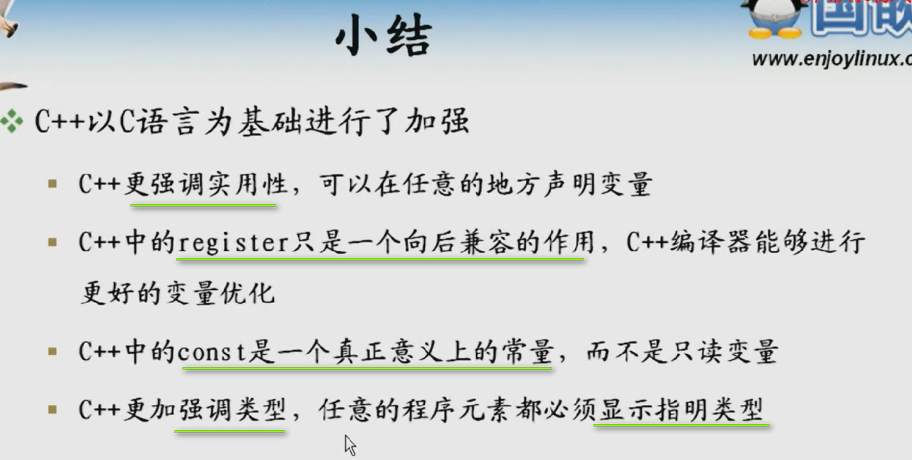
在C++里面 什么都不写



这个就是C++对类型检查的加强

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





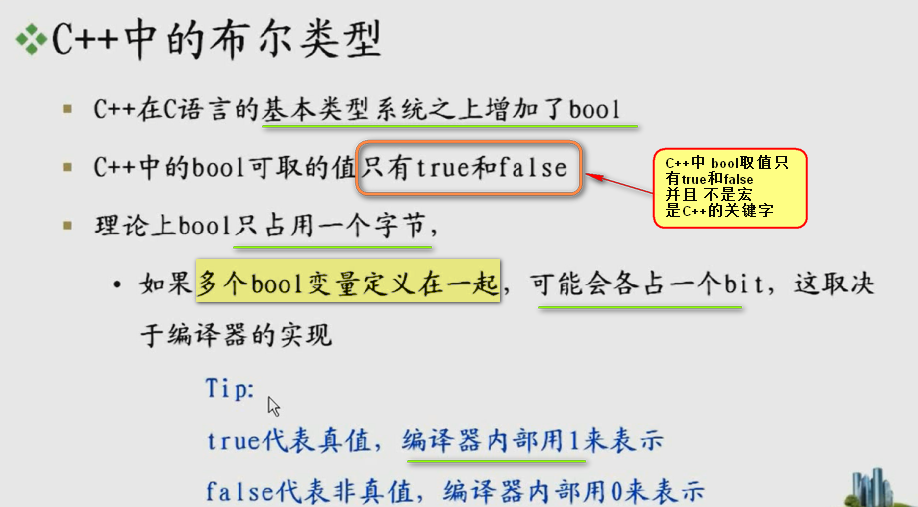
-------------------------------

C中没有布尔类型 喜欢用int代表布尔类型 -----

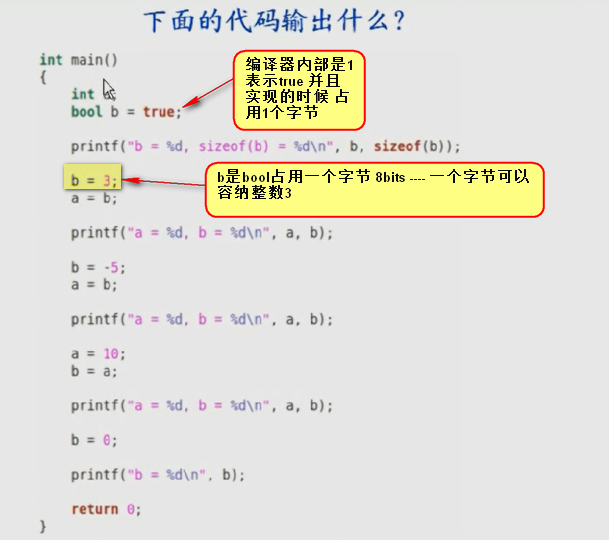
很多函数 返回都是int 但是 只会返回1或者0 这时候1当做布尔类型使用 不像高级语言表现的 所以 C++中出现了bool类型 进入了C++的基本类型系统

由于C++是现代编译器 ---- 有优化

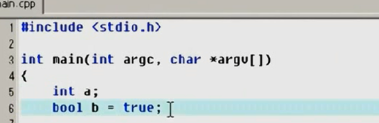
第一行定义了一个布尔类型的变量 第二行 第三行都定义了布尔类型的变量 ---某些C++编译器 就会把这些bool类型变量放到同一个byte --- 可能各占有一个bit

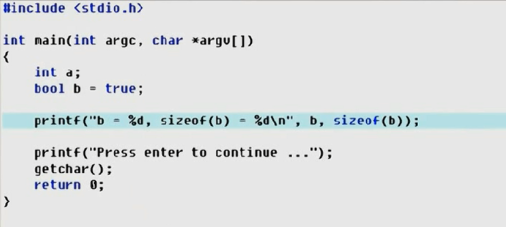


编译器内部 还是用1表示true ---- 这样还是为了兼容C语言 false在编译器内部还是使用0表示

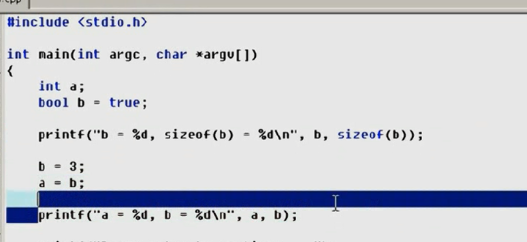


构建一个C++ 和C的工程

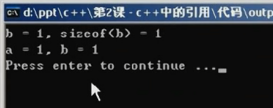
 bool在c++中变成了蓝色 说明已经是c++的一个关键字







按照分析 应该是两个3

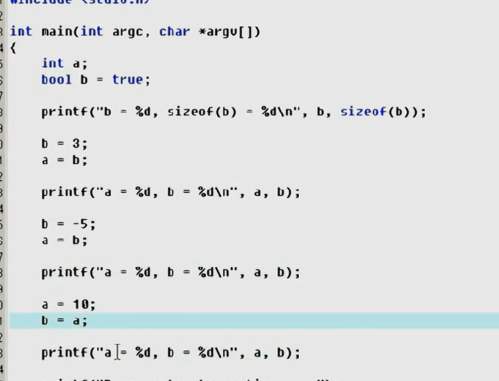
 但是 为什么出现了两个1 ？

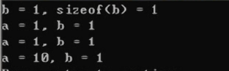
1个字节当然可以毫无压力存放3

重新给b修改为设置为负数 -5

 -----  打印还是1

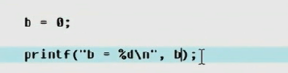
重新修改一下

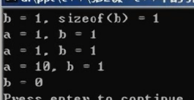




修改的仅仅是a 但是b还是1

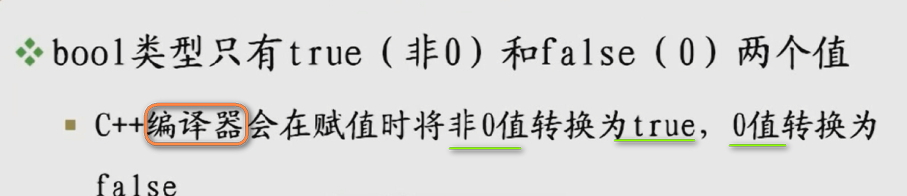
现在直接给b赋值0



 ----- 布尔类型的b就是0

通过这例子 可以得出一点结论

Bool在C++中是一个安全的类型

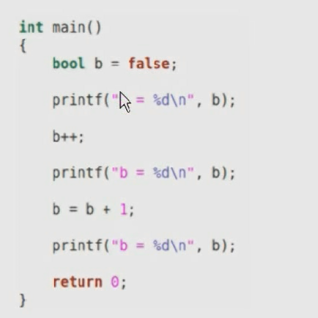


 3非0值 被编译器变成了1 赋值给了b

B=-5 -🡪 -5还是非零 所以 转换成1 赋值给b

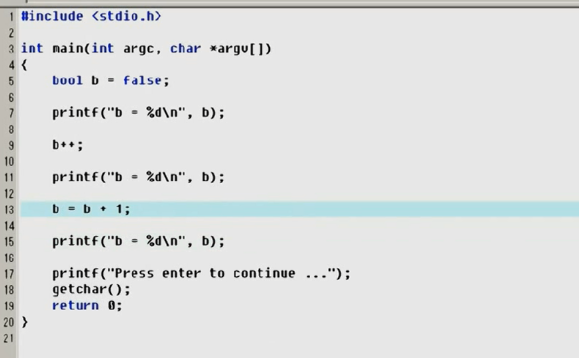
以上都是编译器的行为

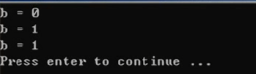
如果bool类型的变量 参与数学运算 会有什么结果？

 false内部表示就是0

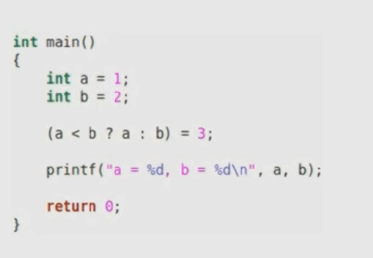
B++ ---- 0+1 就是1 非零 所以 b是1

是否打印出来是0 1 1呢？



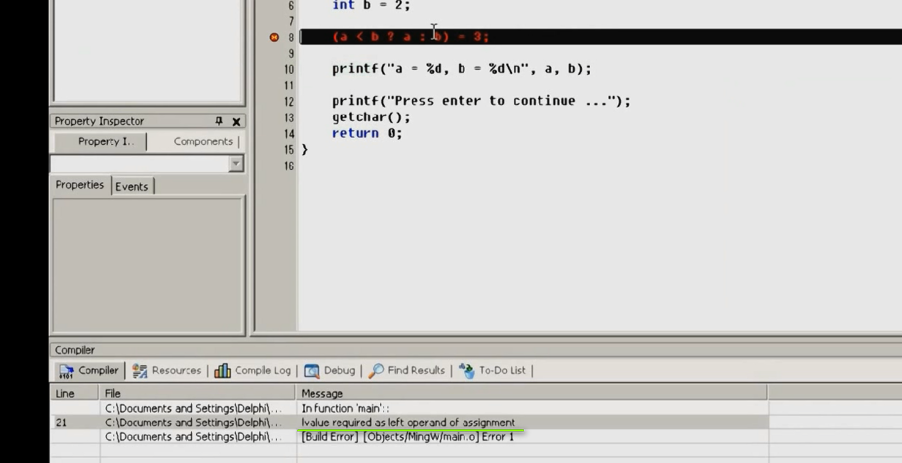


====== 三目运算符的升级



 三目运算符放到了赋值左边 对么？ 因为三目运算符返回的是值 不是变量本身

先到C语言中进行验证



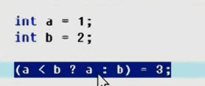
C编译器给出了报错

C语言中不能给值再次赋值

在C++中 编译通过了

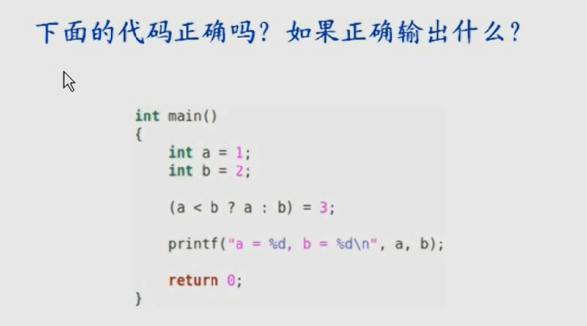
运行





C++ 中 ---- A开始小于b 这样 返回的是变量a 不是变量的值

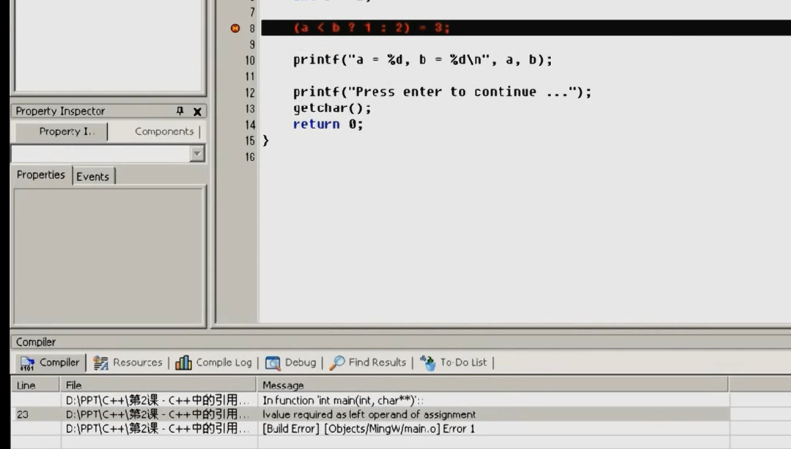
所以 三木运算符 c语言返回的是变量的值 但是c++返回的是变量本身 ---- 所以a的值就是3



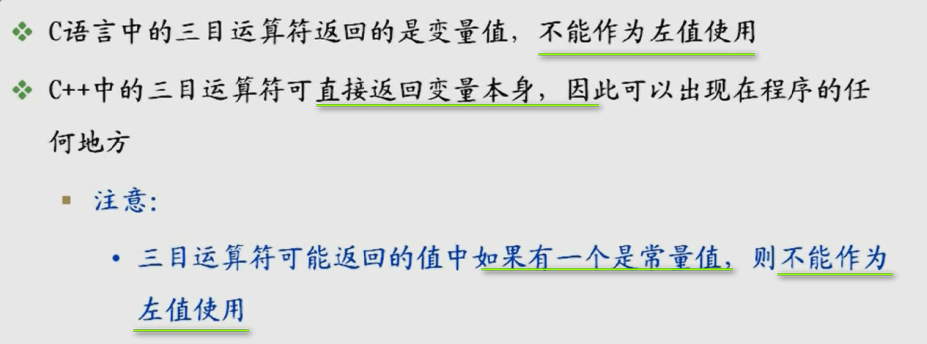
用C语言编译器 编译不通过 用C++编译器 编译通过

代码修改一下

？？？C++编译器能通过么？



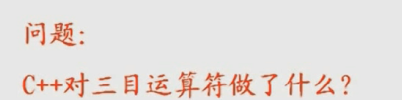
所以 C++也没编译通过





这个返回值 有一个可能是常量值 1 所以 C++编译器不让通过

当且仅当 三木运算符返回值都是变量的时候 c++才会让他返回变量本身



下一节课继续。。。。