第2课 –C++中的引用

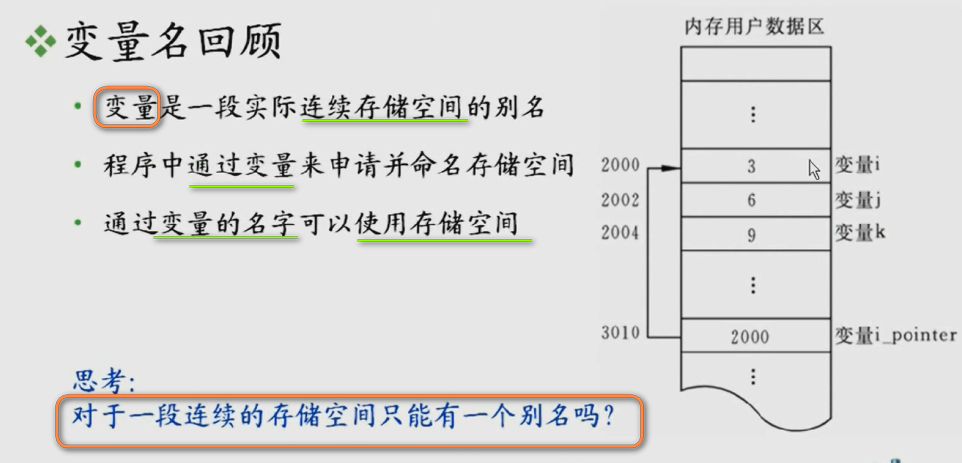


C++究竟对三目运算符做了什么？ 这个就必须引入C++中新的内容 引用 ----

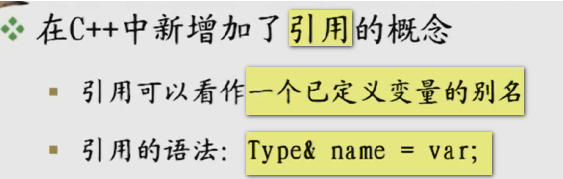
**变量名 就是一段连续存储空间的别名**

**程序里面 对一段连续的内存空间起一个别名 ---- 问题是 只能取一个别名呢？可不可以取多个别名？**

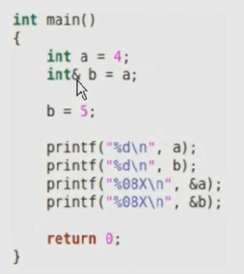
----- 基于这样的假想 C++就提出了引用的概念



比如图中int I =3; ---- 就为变量i分配了从2001内存单元到2001这两个内存单元这块连续的内存空间 变量名就是i

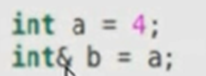


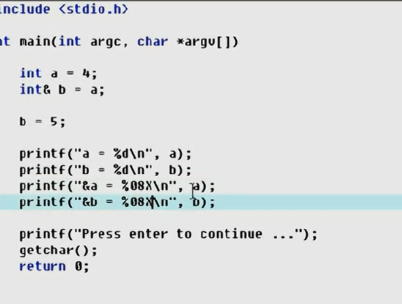
引用的语法看上去指针的定义 ---为了兼容C语言 \*是定义变量的指针 &是定义变量的引用



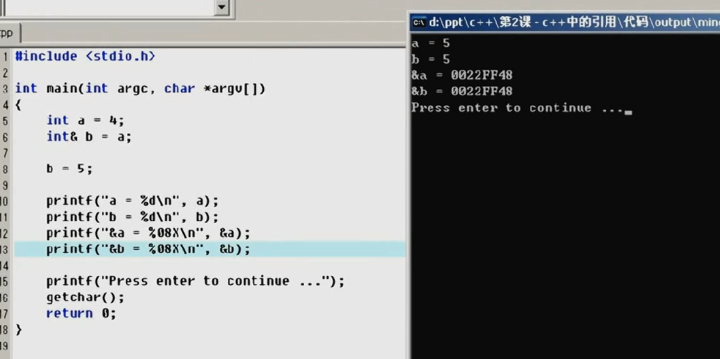
int& b =a; ---- b这个名字就代表了a ----所以 改变了b 其实就是改变了a

a跟b取地址 地址的值一样么？ 概念上说 是一样的 理论上是一样的

 这样b就是a a就是b



编译并运行

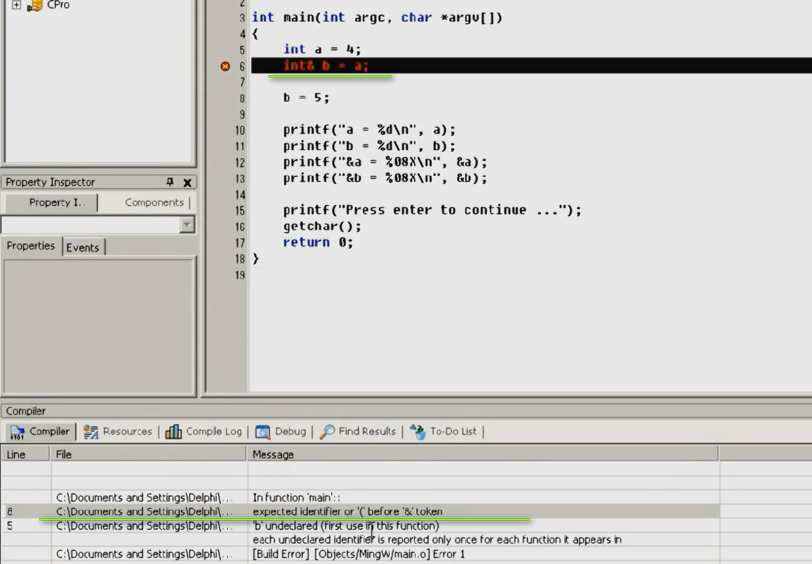


这个程序里面 a没有直接被赋值 为什么从初始的4变成了5？ 因为引用b成为了a的别名 a和b指代了同一段内存空间

从打印a b的地址看出来 ----- a和b指代了同一段内存空间

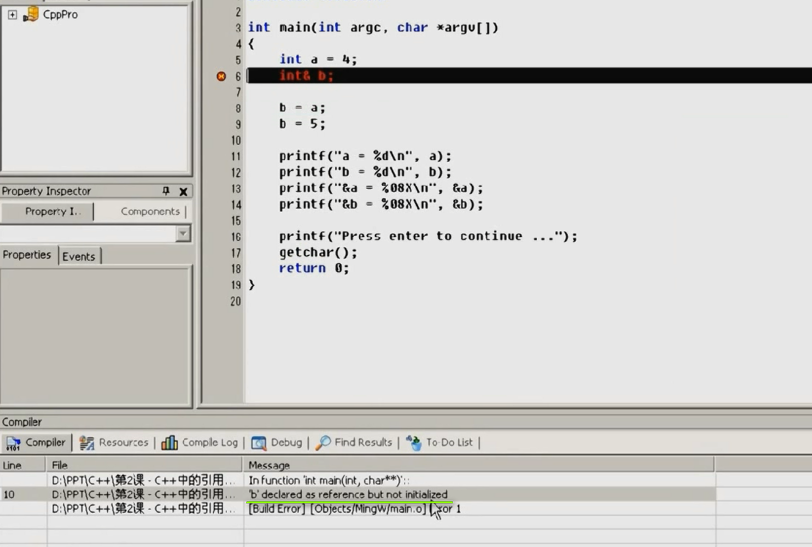
**其实 引用就是定义一个别名**

现在 把这段代码放到C工程中

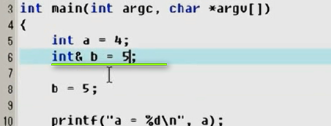
 编译不过 C中没有引用的概念

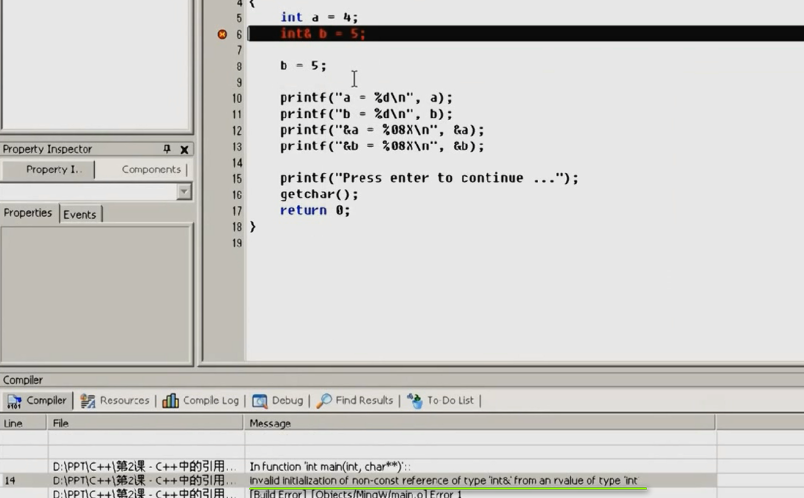


普通的引用 在声明的时候 必须出其他的变量进行初始化 否则就会出错

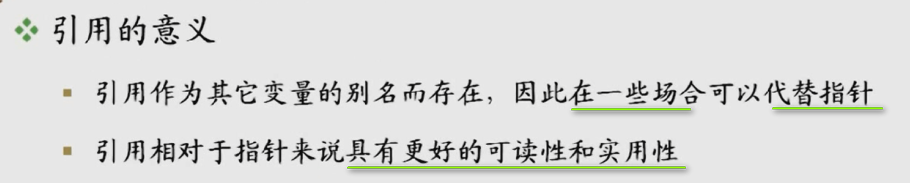
 编译报错 告诉你 声明的引用没有初始化

还有就是 我就不用变量初始化



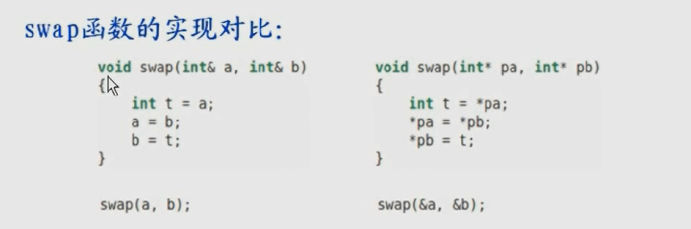


C++的引用存在的意义是什么？为什么要给同一个变量取别名？



C++是一门实用 高效的语言

所以 就会很讲究可读性 实用性 --- 如果做到C++代码好读？比如使用引用这个细节就可以做到这一点



C语言交换函数 一定使用了指针

可读性 -----

看到swap 就知道这个是变量的交换 ----- 但是 传入的实参是a的地址 b的地址 --- 很自然得出下面的结论：swap作用是交换a b的地址 但是 和实际的情况不一样 不看定义 就看错了含义 ---这是C语言有时候 不那么好读的原因

C++中 变成了引用类型的参数 就是其他变量的别名

调用的时候  很清楚 读懂了a b这两个函数的值 比C语言更容易读懂

这个里面有个问题 刚才不是说引用声明 不就是要用其他的变量处理化？ ---- 引用作为函数参数 声明之后 不用立刻初始化【原因 就是 只有函数真的被调用的时候 函数的形参才生效 这时候 形参中的引用才会被声明 并立刻初始化为实参】

【

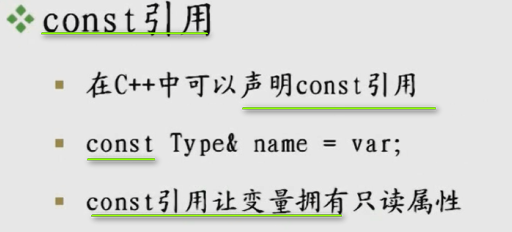
Int a =10;  
 int&b = a; --------- b就是a 都指向同一片内存空间

Int c =a; ----- 这个c就会重新开辟一段内存空间 把a的值10 复制到c这个变量名代表的内存空间中

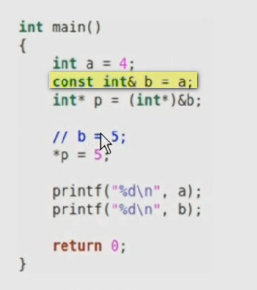
】

如果把引用声明为函数的参数 整个函数就不会像指针那么晦涩难读了

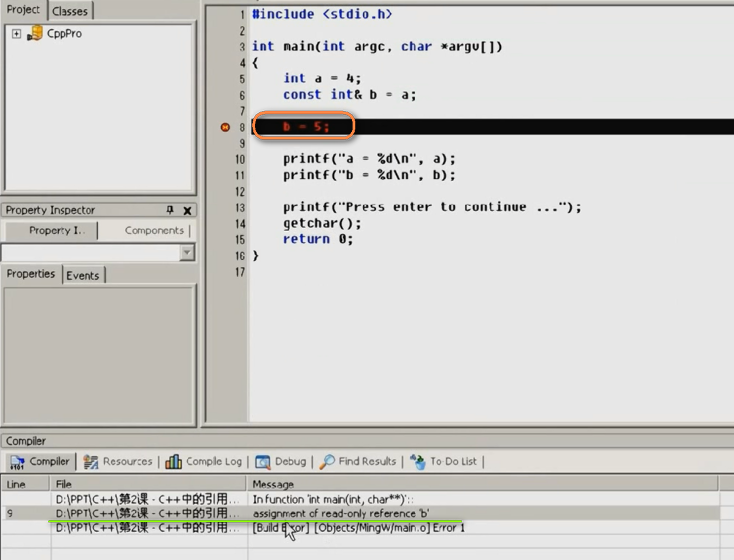
============ const引用



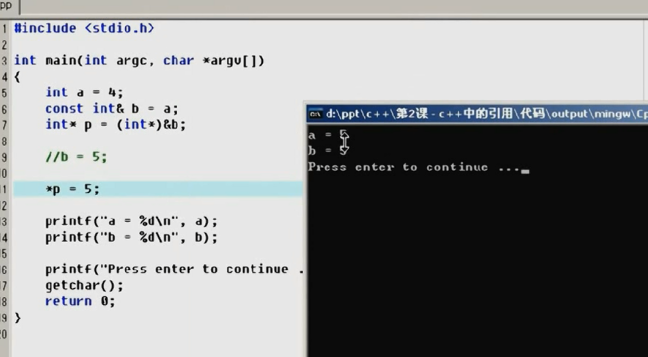
加上了const 就是让一个变量拥有只读属性

 这时候 a通过const int& b把自己变成了一个只读变量 那么 就不能够出现在复制符号的左边 所以这时候 想到使用指针来修改变量a的值

【\*在声明变量 表示变量是指针 &在声明变量 表示变量是引用】



B所引用的变量a就是只读变量 虽然b是a的别名 但是 如果直接使用b 就不能当成左值来使用

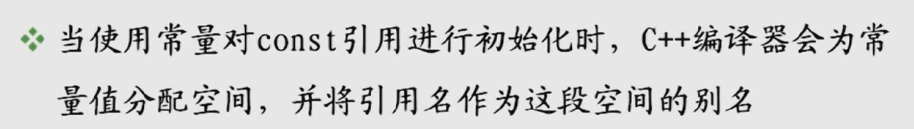




也就是说 **const引用** 让一个变量成为只读变量

Const引用只有这么一点点内容？

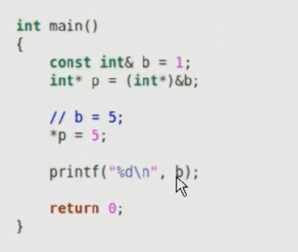
下面这点非常重要！

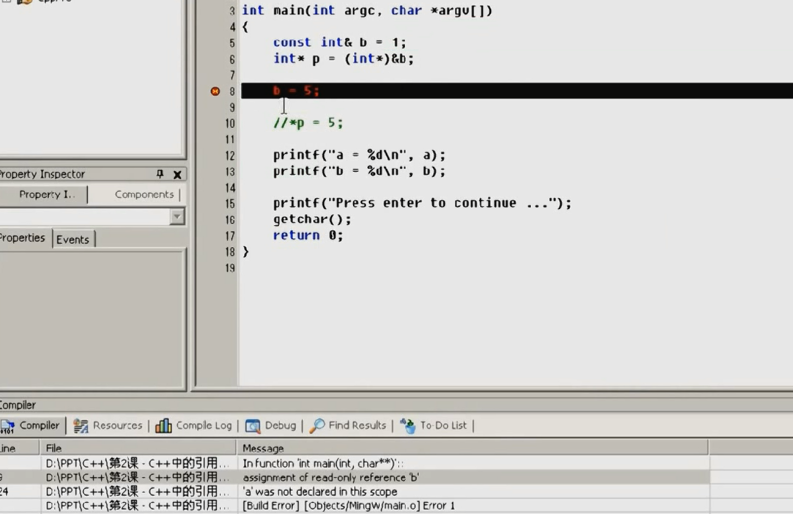


如果程序中  ----- 使用常量值 对常引用进行初始化 --- 此时编译器 就为常量值1分配4个字节的存储空间 原因就是int是四个字节的存储空间

分配了4个字节的空间 ---- 此时编译器在分配好空间之后 使用b作为这4个空间的别名 ---- 是这样么？



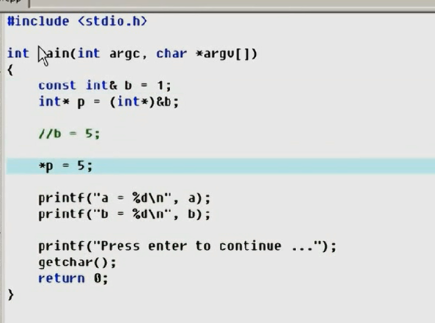




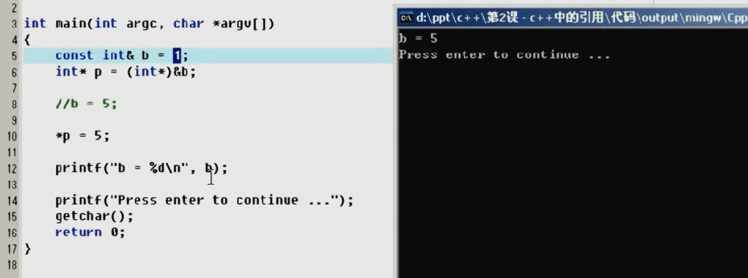
**前面没出错 意味着可以使用常量对常引用进行初始化 但是 前面也说了 不能使用常量对普通引用初始化**

 b是一个常引用 意味着只读变量 没有其他的变量

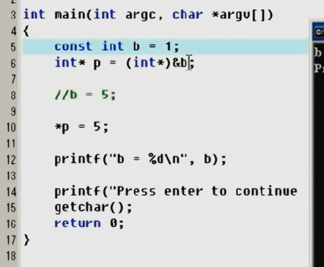
【注意 这种int& b=1; 就会编译不过 但是 加上了const 就会通过 常引用的特点】



编译通过 意味着取b的地址 给指针p

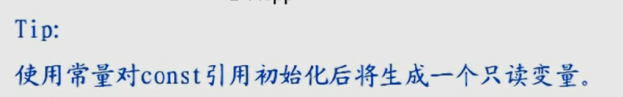


**这样证明了 得到的是绝对的是只读变量 而不是const常量 ---- 因为得到的b 赋值的时候 跟着改变了 这个不是常量的行为**



常量值 无论怎么搞 编译器都从符号表中 获取这个常量的值

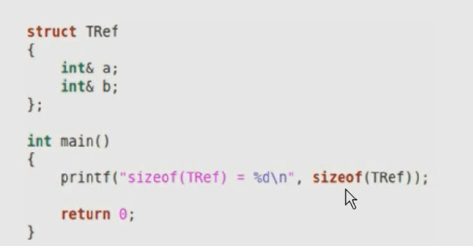
但是 常引用就不一样



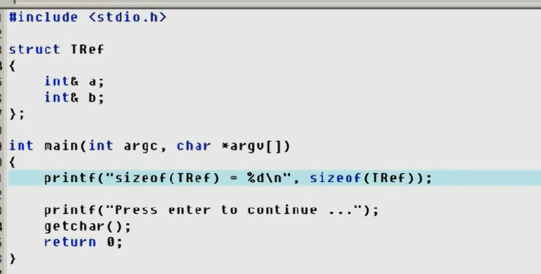
使用常量对const引用初始化之后 生成一个只读变量 只有一个名字 就是引用的名字 ---- 这个只读变量和C语言中的只读变量一个意思 没有区别



按照刚才的说法 引用都是变量的别名 要么只读变量的别名 ---- 那么 按理说不该占用空间



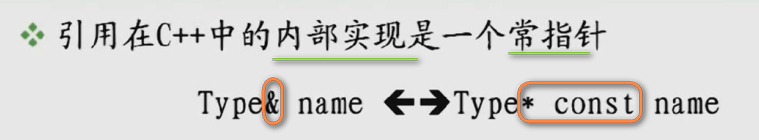
假设 引用没有自己的存储空间 那么 这个输出就是0 不占用空间



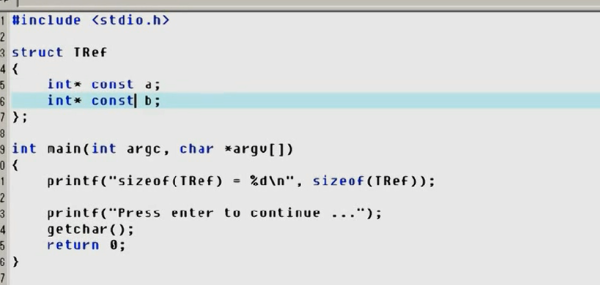


很不幸 打印出来了8 不是0

为什么会这样呢？



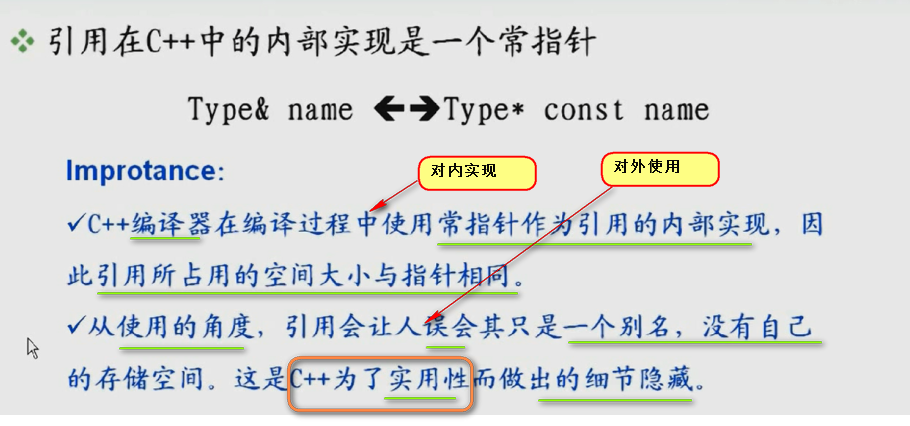
所以 这就解释了 这个struct大小是8 不是0



做一个等价代换



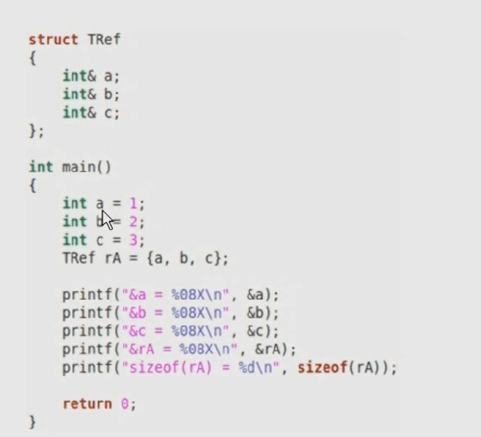
引用就是有自己的存储空间 并且每个引用存储空间应该在32为机器是4个字节 否则是8个字节

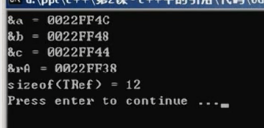


所以 引用的出现 还是实用性的体现 ---- 下面



**这样 引用使用的非常方便 不用\*来使用 实用性 可读性在C++中都有提高 --- 用起来非常方便**



 a b c地址分别相差4

rA的地址和a b c地址似乎有差别

看看c的地址 和rA 的地址相差了多少个字节？

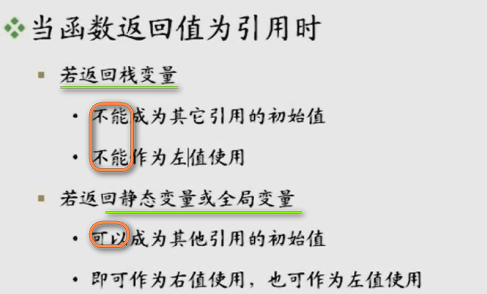
a b c有各自的存储空间 并且在内存中是相邻排放的

4c -48 -44

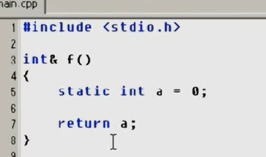
rA里面的三个a b c引用有自己的存储空间 --- 在C++内部实现 是指针而已 所以rA要占用12字节的内存空间

【引用用起来好用】

前面讨论了函数参数是引用的时候 意义是什么



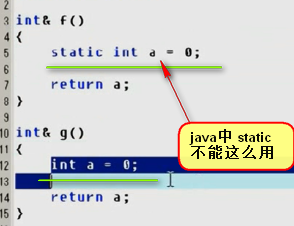
引用返回值示例

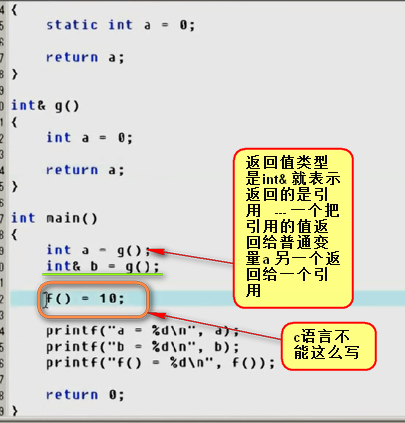


f返回值是返回引用

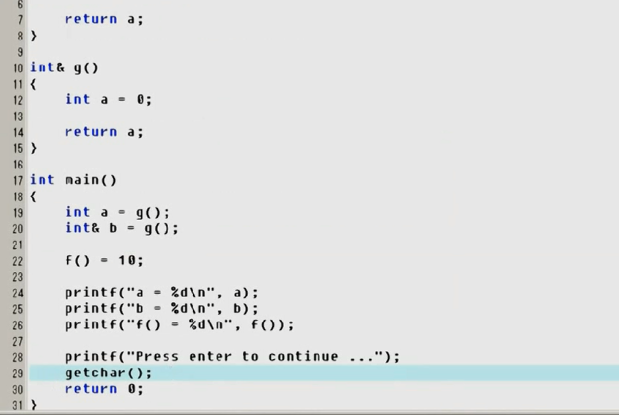
函数体中 直接返回一个静态的局部变量

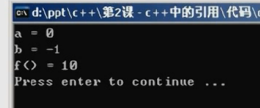
【再次体现出 引用的实参传参和 函数体内部实现 都是非常好用的 返回一个局部变量 就相当于java中 子类返回给父类 假设返回值是 int& c ---- 这个return a;就是 int&c =a;】





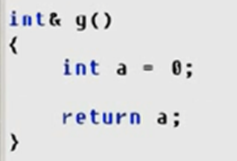
但是 f()=10; ---- 因为这个里面 a是静态的局部变量 所以 可以作为左值 所以 就是这个f()中的a就变化了





a --- g()返回一个局部变量 这个局部变量的值赋值给a

b代表了g中的局部变量

 但是 这个局部变量a在g()被调用完之后 就被释放掉了 这样 a所占用的内存空间也被释放掉了

**这个时候 引用b是已经被释放了的内容 ---- 所以 里面是随机值 ------ 所以 为什么给这个局部函数取一个别名？ ----- 这个就跟返回一个局部变量的地址一样**



由于 我们f()里面返回的静态局部变量 ----- 函数调用结束之后 不会销毁对应的static的内存空间 那么这个返回的就是 a这个静态局部变量的引用 ----- 因为空间没有被释放 就可以当做左值来使用 ----- 因为这 f() 返回了静态局部变量a的别名 所以 f()=10 就是给这个a赋值为10

充分印证了

\*\*返回了引用的函数里面 ---- 返回的是栈变量 就是局部变量 那么 就不能成为其他引用的初始值 也不能作为左值使用【内部局部变量占用的空间被释放了】

上面的-1怎么来的？

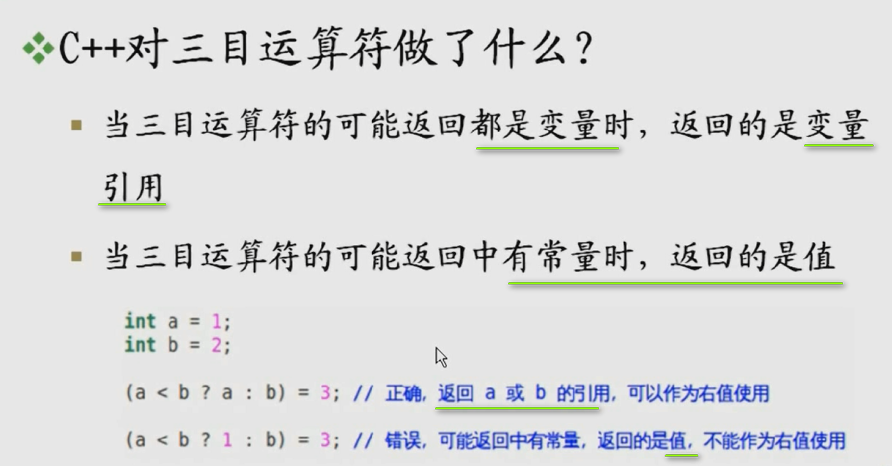


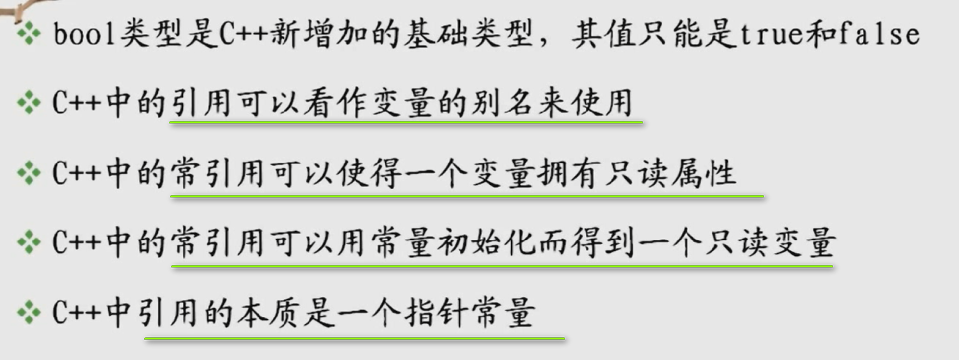
其实在编译的时候 c++已经建议我们不要这么做 --- 说的是局部变量a被返回了引用 warning

返回了之后 局部变量的引用 作为了其他引用的初始值  ---人家c++编译器给你警告了 你还要这么做 没办法了

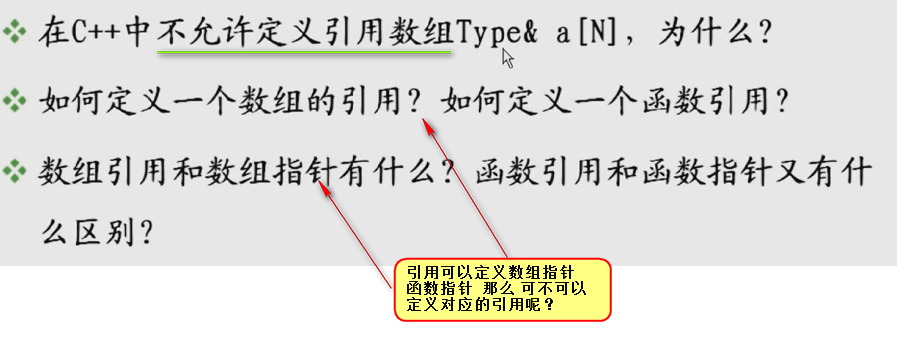
如果是静态变量 或者全局变量 就没关系了

**--引用其实本质上就是指针**





思考题



**【引用的实质是指针 也就是 Type& ref ----- Type\* const ref ------ 根据左数右指的规则知道 这个Type\* const ref 是表示ref这个指针是常量 ---- 也就是这个指针本身是常量 ----- 也就是这个是常指针 ----- 这样的意思就是引用一声明出来 初始化之后 就永远指向同一个内存空间 ---- 那么这样 这个常指针 也就是引用 不就和这段内存空间对应的变量名一样了么 都专属于同一片内存空间 ------ 所以 引用和变量名此时的作用就变得一致了 都是专属于同一片内存空间 ！！！ 所以 C++干脆 把这种常指针就变成了对应的内存空间的别名 ------ 所以就把引用看成别名】**