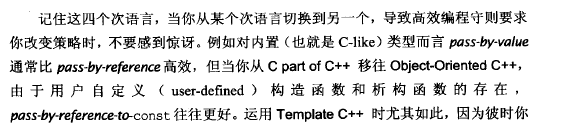
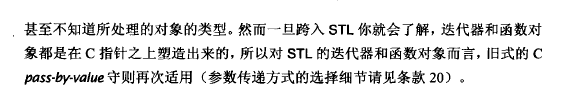


所以如果是引用的话~ Widget& w,则是调用Copy Assignement？就会相当于引用了。

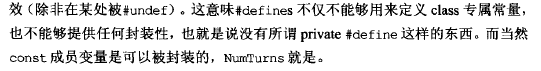


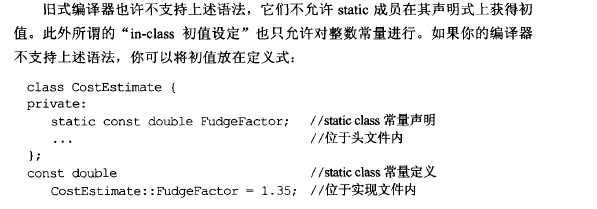


Const Widget& widget

## 用const等替换#define



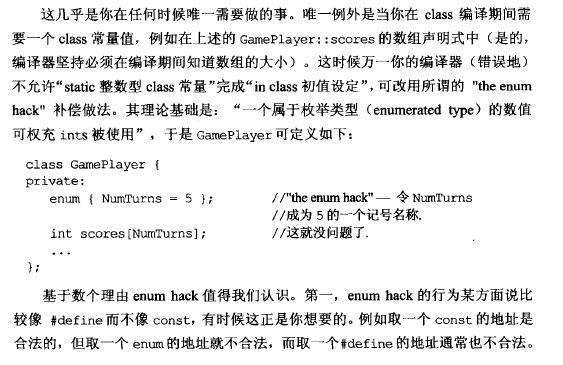




上述语法说的是，直接在类中声明的时候，就把值定了。

Static const double FudgetFactor = 1.35; 在CPP中，声明一下，不设初始值。

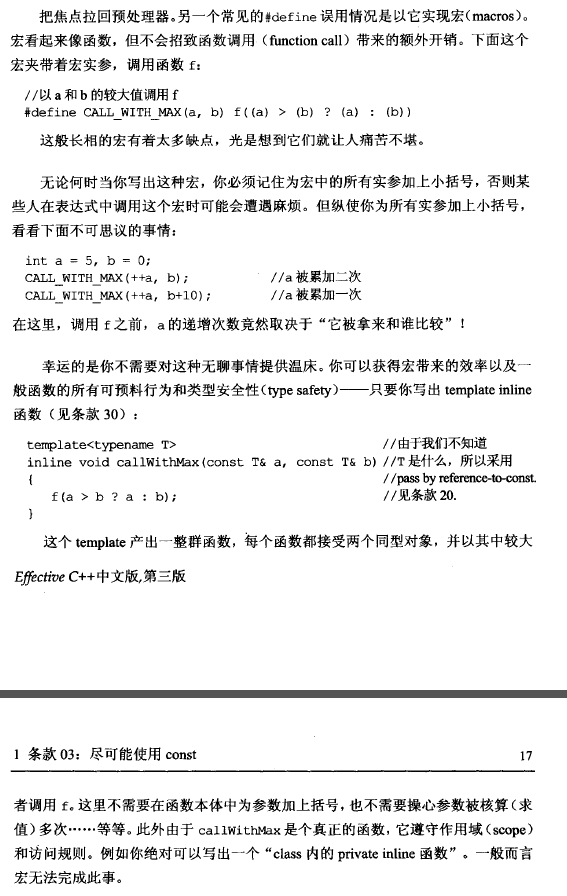
Const double CostEstimate::Fu…..

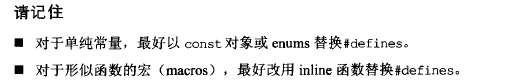


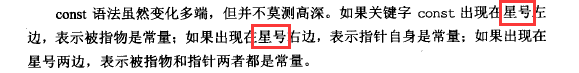
需要常量值，但是老编译器又不能通过static const的直接赋值时，

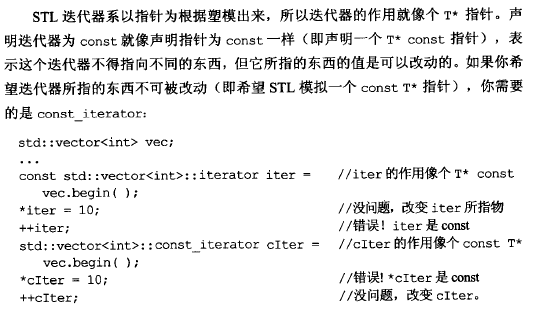


这个很有意思，用宏的话，可能会造成重复调用累加多次，当然有些函数也有可能，



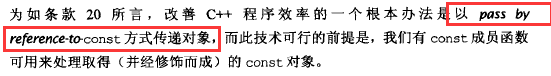




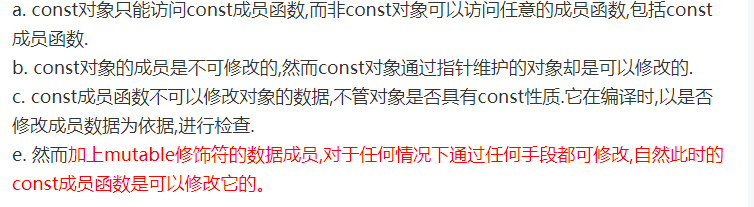


所以const 的迭代器，不能++,const\_iterator的迭代器，不能修改指向的值.

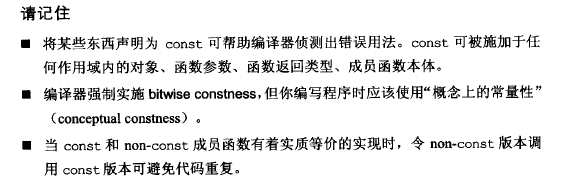
const 放在函数后面，成员函数，



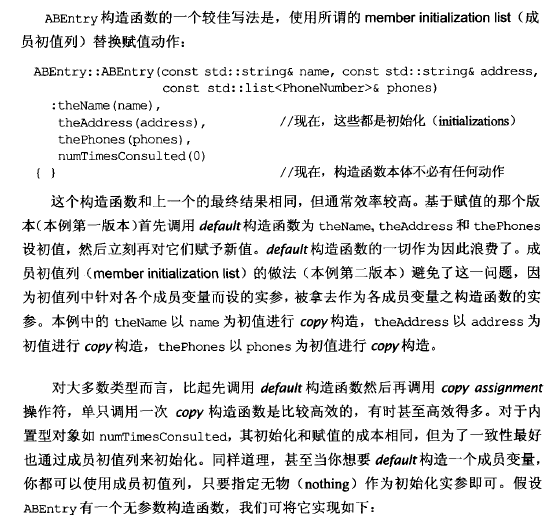


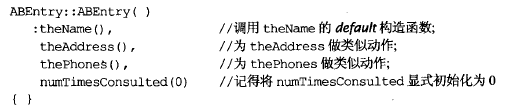


对const成员函数，可以通过指针去修改！！！(有空的话，去测试一下EventSystem里面的GetDesignSize还是啥的函数需要Size，size可以从Draw还是Arrange，是一个const的函数中获得，我想从中输出值却一直没办法，现在看来，可以用指针或是mutable！)



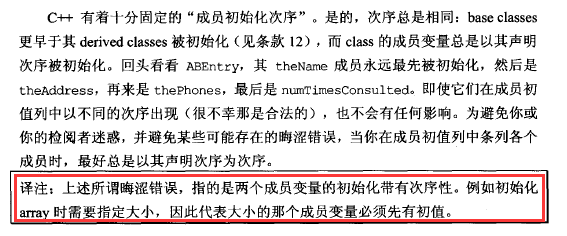
## 确定对象使用前已初始化





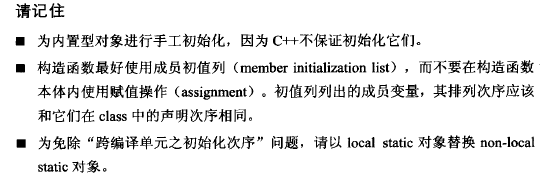
避免在构造函数中，使用theName = name这种，这样的是赋值，而不是初始化。

const 或是 references ，一定要初值，不能被赋值，所以最好总是使用成员初值列。



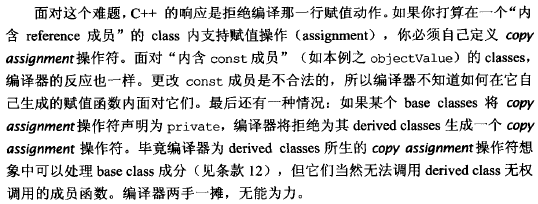
!!!这个很重要呀~用上一个声明来初始化下一个的数组这样的。





最后一条的意思，就很像是单例了~

# 构造，析构和赋值运算（64页）



总结一下其实是这个情况，就是如果你没有手动写，而是调用类中默认的copy assignment操作符的话，他会拒绝为string& nameValue这样的reference变量以及const T objectValue

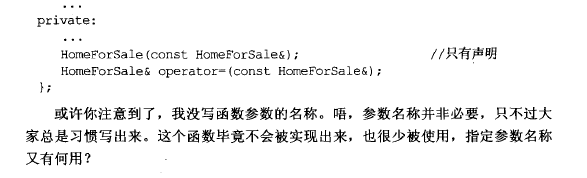
这样的const对象赋值。



## 若不想使用编译器自动生成的函数



你不声明的话，如果调用到，编译器也会帮你加上,方案1，自己进行显示声明为private，不够安全。

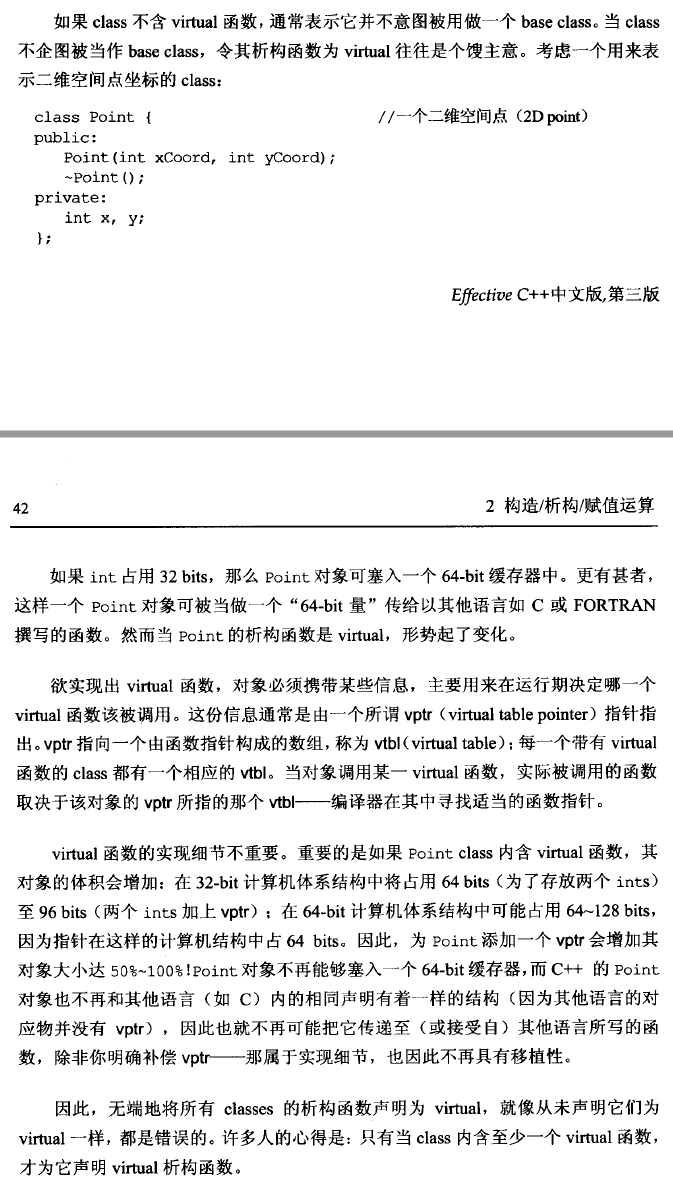


参数名称不是必要的！

方案2：用一个类，然后再去继承他,可能会多重继承就是了

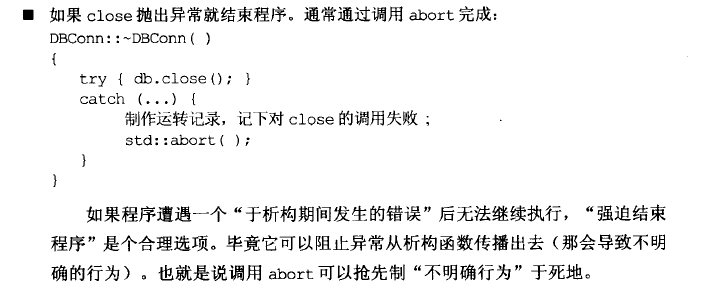
## （可记录）为多态基类声明virtual析构函数70页

这个真的非常有用！！积累的析构函数，声明virtual ~XXX()，这样，哪怕取的是基类的指针指向子类的对象，也能够在清除的时候正常调用子类的析构进而清除所有！



!!!！原来是这样，声明virtual 会导致该对象多携带一个VPTR指针！！原来没有virtual声明的话，本来可以占用很小的内存，但是声明之后呵呵！而且还会影响移植性！！！别的语言没有VPTR指针

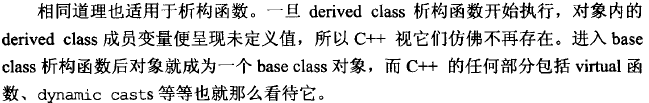
## 别让异常逃离析构函数



## （可记录）绝不要在构造函数和析构函数中调用virtual 函数！！！！



会被调用的是基类的！！构造函数先调用基类的构造函数，那么理所当然的，继承类都还没有初始化，运行起来肯定会有问题，所以会被编译器视而不见



最关键的在于，我们常常会使用Init函数来在构造函数中初始化，但是如果此时在Init中包含了virtual的函数，那么。。。。。这个还真的很容易用错，关键是不会引发编译器报错！在UE4中试一下呢~UC++ 此到第82页

# 资源管理(91)

可以在某段函数内使用智能指针， auto\_ptr，则在离开该函数时，会自动销毁并删除它所指向的物体，所以不要让多个智能指针指向同一个变量。

