### 导读：

1. 默认构造函数是一个可被调用而不带任何实参，它要么没有参数，要么就是每个参数都有缺省值
2. 拷贝构造和拷贝赋值的区别：当一个新对象别定义时，就一定有个拷贝函数被调用，不可能调用赋值操作

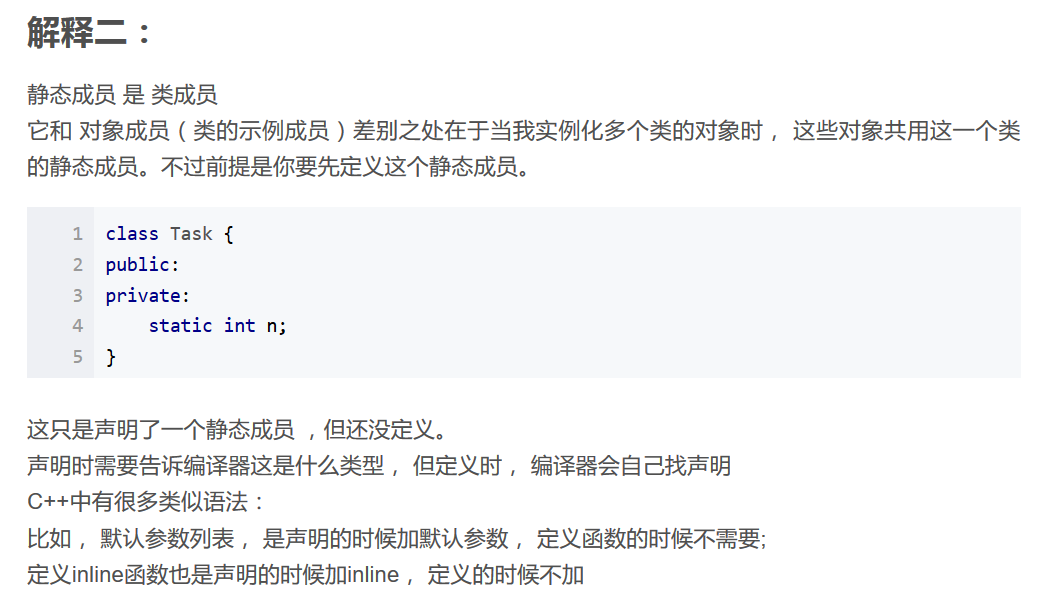
### 条款1：视c++为一个语言联邦

1. c语言没有重载！！！

### 条款2：尽量以const,enum,inline替换#define

1. 对于类中定义static const int Numturn=5的解读，由于是static为了保证所有该类定义的对象只有一份该成员，所以只能把类中当做它的声明，同时需要在类外提供一份单独的定义，并且在定义时，不需要加static，因为加了static按照c语言的语言，那么该成员只能在当前文件有效，会导致一些问题，而且加static是没有必要的，在定义的时候会指示类的限定符，可以找到对应的声明





B.#define定义的东西没有作用域概念，即不区分private,public,protected,定义之后的所有地方都能用

C.enum行为更像#define不像const，但是定义一个enum会保存一个记号标记。不能取enum，#define的地址，可以取const地址

D.对于单纯常量，用const或enums替换#define;对于形似函数的宏，用inline替换#define

### 条款3：尽可能使用const

1. const iterator相当于T\* const，const\_iterator相当于const T\*
2. const如果作用于函数上（即声明常量函数，而不是在返回值或参数上），可以作为函数重载，在对象调用时，常量对象调用常量成员函数，普通对象调用普通成员函数
3. mutable类型声明的作用，可以使得成员变量在const函数中仍可以被改变

### 条款4：确定对象被使用前已先被初始化

1. 内置类型对象如int,有可能初始化和赋值的成本一样高，因为它不一定保证能在复制动作之前获得初值，
2. 对于类的初始化列表，它的成员变量总是以其声明次序被初始化
3. 由于每个编译单元全局的static变量的初始化时机无法确定，如果一个变量用到未初始化的静态变量，导致出错，所以可将静态变量放在函数中，然后返回该变量的引用，需要使用该静态变量时使用该函数即可。如

FileSystem& tfs(){

Static FileSystem fs;

Return fs;

}

### 条款5：了解c++默默编写并调用哪些函数

1. 对于一个类，编译器会保证基本的四个函数，构造函数，拷贝构造函数，拷贝赋值函数，析构函数，若没有编写前面的四个函数，编译器会创建默认的
2. 对于拷贝赋值函数，如果成员变量中有引用或者常量，如果有赋值语句的话，编译器会拒绝编译这行语句（即编译会将拷贝赋值函数标记为删除）
3. 如果某个基类将拷贝赋值语句声明为私有，那么在派生类中编译器会拒绝生成拷贝赋值函数

### 条款6：若不想使用编译器自动生成的函数，就该明确拒绝

1. 若不想对象不能拷贝赋值和拷贝构造，那么就把拷贝构造函数和拷贝赋值函数声明为private（编译时出错）并且不定义（成员函数或者友元函数使用时链接时出错）

B.若想将链接时错误提前到编译时，可生成一个Uncopyable的基类（两个函数声明为private并且不定义），使用时继承该基类

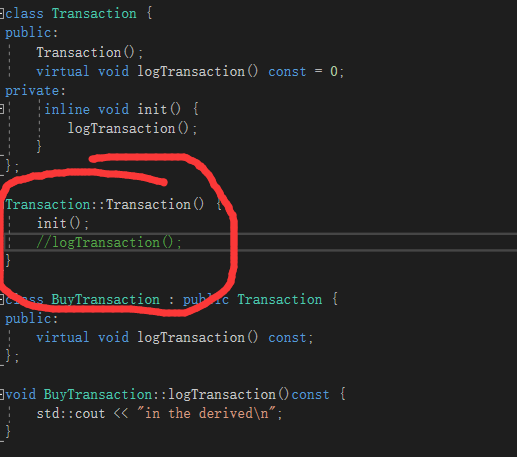
### 条款7：为多态基类声明virtual析构函数

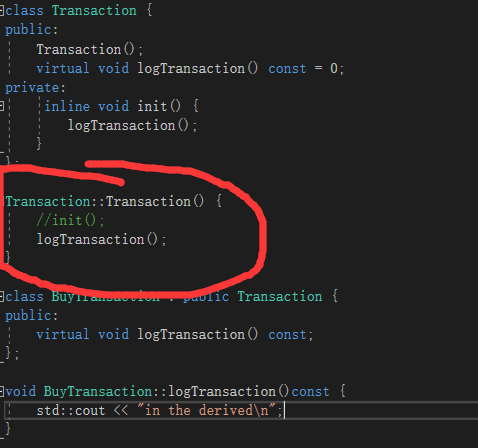
1. 当一个class里没有virtual函数，通常不会被用作基类，可采用继承的方式

### 条款8：别让异常逃离析构函数

### 条款9：绝不在构造和析构过程中调用virtual函数

1. 在基类的构造函数中调用virtual函数，如果你定义了一个子类的对象，在基类的构造函数执行时，只会调用基类对应的虚函数，是因为这个时候子类的成员还没被构造出来（由于子类的虚函数有可能会使用这些对象），所以安全的做法是在子类对象的基类构造期间对象的类型是base class而不是derived class（同理对于析构函数，在基类调用的虚函数是基类的虚函数，这个时候子类的成员变量已经被析构了）
2. 下面两种方式的区别，直接调用纯虚函数，和外面加一层函数包裹（init,再调用纯虚函数），实例均定义一个子类的对象





第一种编译链接没错，运行出错；第二种链接出错

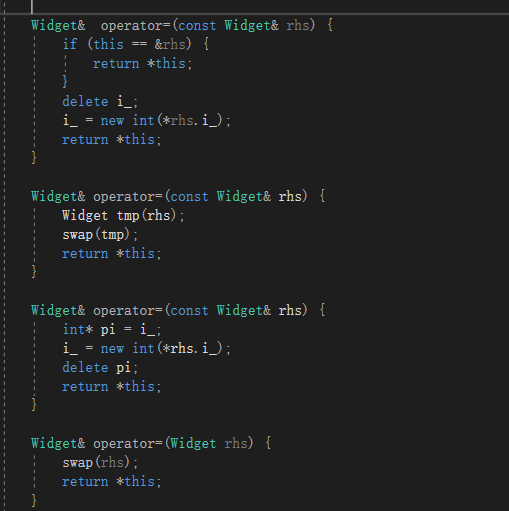
原因是加了一层函数包裹，两个方面语法均成立，第一在构造函数里可以调用init函数，第二可以在init函数中调用纯虚函数（符合多态的性质），所在链接时没问题，运行时找不到实际函数地址，故出错

### 条款10：令operator=返回一个reference to \*this

1. 这只是个协议，并无强制性，不遵循代码也可以运行，但标准库都用了这个标准

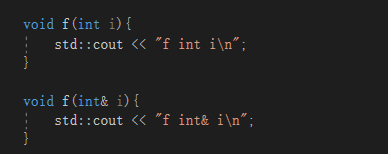
### 条款11：在operator=中处理“自我赋值”

1. 使用copy and swap 技术，四种处理自我赋值（只有当对象中有指针的时候需要考虑），最后一种写的最漂亮



### 条款12：赋值对象时勿忘其每个成分

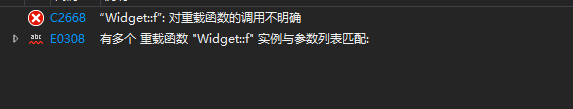
1. 各种重载函数不同写法，如类中有如下成员函数（1）



使用 对象.f(1)，会调用第一个函数，因为临时变量不能使用引用，即它不是左值，同时由于i是常量（注：以上任意一个成员函数的参数加上const，那么就可以运行了）

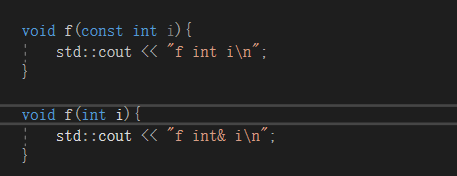
int i = 9;

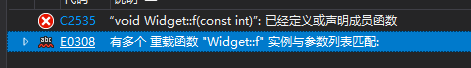
然后使用 对象.f(i)会报错



由于i对于两个函数都是匹配的

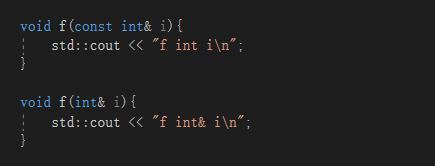
（2）





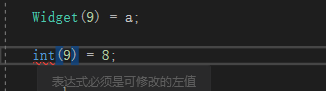
传入变量，临时变量都不行，没有达到重载，因为传值加const，即使内部能改变，也影响不到外面的值，所以不当做重载

而对于



因为加了const对引用会有影响，两种表现的方式不同，一个值可以变一个值不能变，所以可视为重载

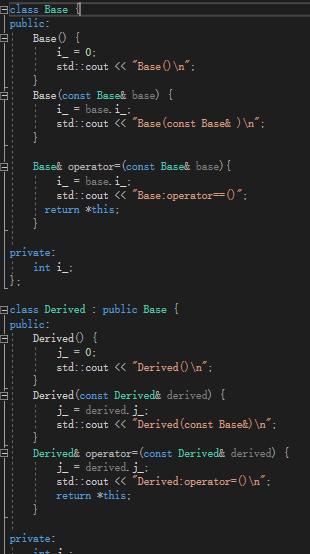
B.临时变量是右值



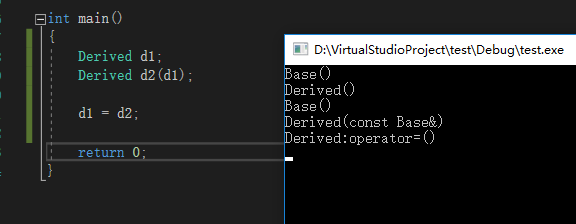
但是一个能编译成功的原因是：Widget定义了拷贝赋值函数

C.注意，如果自己写了拷贝赋值和拷贝构造函数，那么在子类的拷贝构造和拷贝赋值中需要显示的调用基类的拷贝构造和拷贝赋值，

代码如下

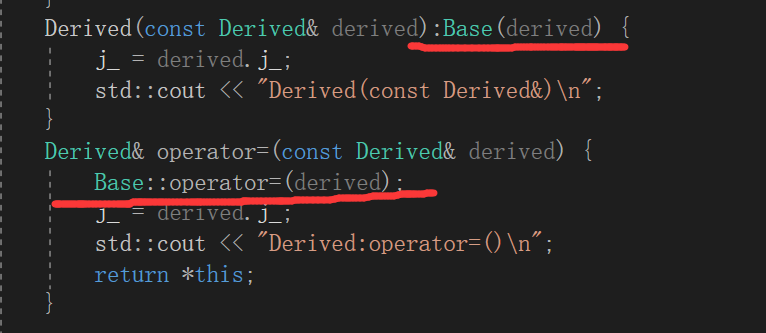


不然调用情况如下



对于子类，基类的默认构造函数没有影响，但是拷贝构造函数会调用基类的默认构造函数，对于拷贝赋值即不会调用基类的任何代码

修改如下



结果：



D.不能用拷贝赋值函数调用拷贝拷贝构造函数，试图构建一个已存在的对象，然而用拷贝构造函数调用拷贝赋值函数是可以的（但是不建议），是因为在构造函数（无论是默认构造还是拷贝构造）中会多做一些操作，如为一个对象分配空间

E.让拷贝构造和拷贝赋值共享代码的方法，声明一个private的init方法，放入共同的代码

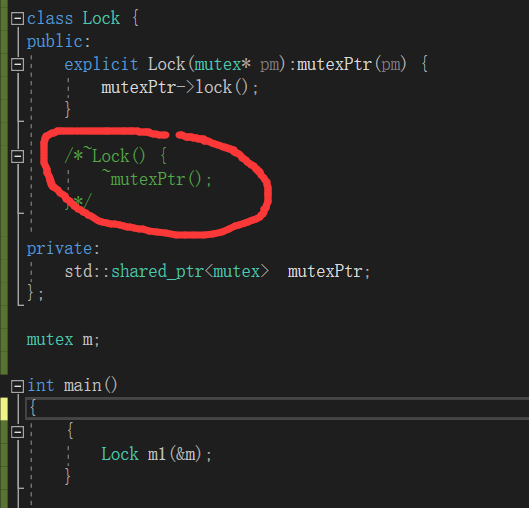
### 条款13：以对象管理资源

1. 使用auto\_ptr的对象，在赋值或者拷贝构造过程中，那么赋值那方会变成null，只会存在一份指针指向资源

### 条款14：在资源管理类中小心coping行为

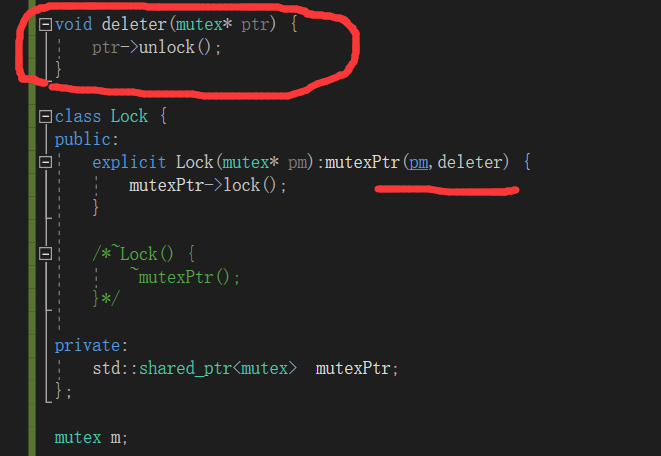
1. 对于某些资源不是heap-based的，智能指针不适合作为资源管理者，像mutex,因为在智能指针中析构函数中调用的delete，而mutex需要的是unlock

B.



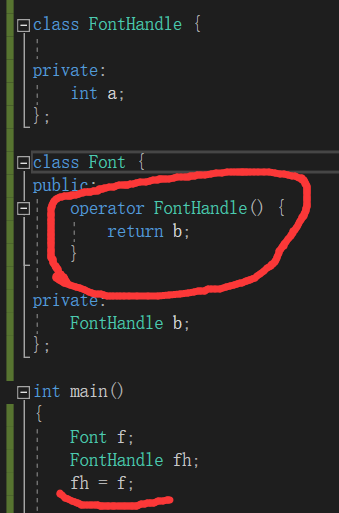
运行上述代码会出现出错，原因如下，m1出了作用域范围，即会调用析构函数，由于没有显示定义，所以会默认生成，即为红框内代码，由于mutexPtr是智能指针但是没有其他的对象引用，所以调用mutex的析构函数，但是由于传入的m是个全局对象，现在还不能被析构，所以出现问题

将代码改为如下形式即可，定义一个智能指针的删除器



### 条款15：在资源管理类中提供对原始资源的访问

1. 无法将同保存同类型的智能指针赋值给对应类型的指针，对象无法给指针赋值，没有适当的转换（不过可调用智能指针的get成员函数实现）
2. 隐式转换函数可实现不同类型的转换，但事实上这种转换十分的危险，可能在需要Font对象时创建一个FontHandle对象



### 条款16：成对使用new和delete时要采取相同形式

1. 单一对象的内存布局和对象数组的内存布局一般不一样，数组的内存通常还包括“数组大小的记录”

### 条款17： 以独立语句将newed对象置入智能指针

### 条款18：让接口容易被正确使用，不易被误用

### 条款19：设计class犹如设计type

1. c++就像在其他oop语言一样，当你定义一个新的class也就定义了一个新的type

### 条款20：宁以pass-by-reference-to-const替换pass-by-value

A.

### 条款21：必须返回对象时，别妄想返回其reference

1. 友元函数声明在类中的话，不受public,private,protected的约束，声明在类中的目的是为了访问类中的私有成员

### 条款22：将成员变量声明为private

1. 某些东西的封装性与“当其内容改变时可能造成的代码破坏量”成反比
2. 从封装的角度看，其实只有两种访问权限，private(提供封装)和其他（不提供封装），protected不比public更具有封装性

条款23：宁以non-member,non-friend替换member函数

1. non-member、non-friend函数比起member函数来说，它会增加更多的可访问private成员变量的函数，使得封装性变差
2. namespace和class不同，前者可以跨越多个源代码而后者不能，意思为namespace std可以声明在<vector> <list>等这些头文件中

### 条款24：若所有参数皆需类型转换，请为此采用non-member函数

1. 只有参数被列于参数列内，这个参数才是隐式转换的合格参与者

如：class Rational;//声明operator\*为成员函数,同时有默认参数的构造函数

Rational a;

Rational result = a \* 2;//成功

Rational result = 2 \* a;//错误

第二个2没有在参数列表中，无法进行隐式转换

### 条款25：考虑写出一个不抛异常的swap函数

### 条款26：尽可能延后变量定义式的出现时间

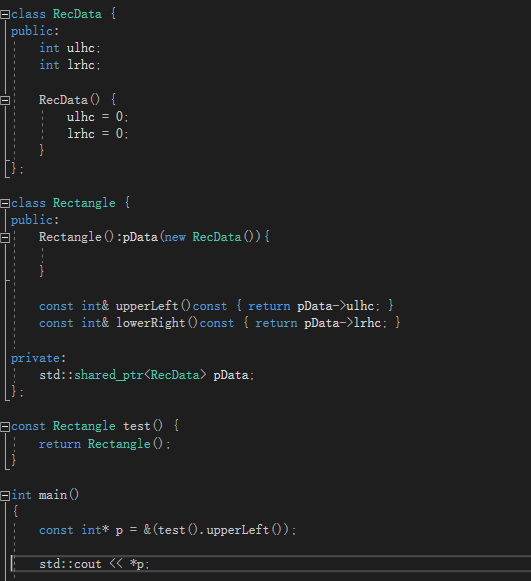
A.你不只应该延后变量的定义，直到非得使用该变量的前一刻为止，甚至应该尝试延后这份定义直到能够给它初值实参为止。如果这样，不仅能够避免构造和析构非必要对象，还可以避免无意义的default构造行为。

### 条款27：尽量少做转型动作

1. 使用四种类型转换会导致生成一个临时变量，如果其中想对对象进行修改，那么影响不到原对象。
2. 在使用dynamic\_cast的时候，可以考虑用虚函数替代

### 条款28：避免返回handles指向对象内部成分

1. 临时变量的语句执行完毕之后，临时变量便被销毁，如下，test函数生产的临时变量会在当前语句执行完后销毁，导致p指针成为一个空悬的指针，所以应该避免返回指向对象内部的指针或引用



1. 在某些情况下需要成员函数返回数据成员的handle,像vector中的operator[]就是返回reference指向容器内的元素

条款29：为“异常安全”而努力是值得的

1. 为代码考虑异常安全性。首先是“以对象管理资源”，那可以阻止资源泄露。然后是挑选三个“异常安全保证”中的某一个实施于你所写的每一个函数身上

### 条款30：透彻了解inlining的里里外外

1. 定义于class内部的函数被认为是inline函数
2. inline函数通常放在头文件（也有例外）中，因为inlining在大多数c++程序中是编译器行为
3. 编译器有递归或循环的函数inlining,而对所有的virtual函数也会使inling落空（因为vritaul是在运行时在知道函数的本体）
4. 如果取了inline函数的地址，那么编译器会为这个inline函数生成一份函数代码
5. 不方便对声明为inline的函数进行升级，因为inline函数代码已经编进所有代码里了，需要全部重新编译
6. 大部分调试器不能在inline函数中设置断点

### 条款31：将文件间的编译依存关系降至最低

a.c++并没有将“接口从实现中分离”做的很好，在类中声明的成员函数（未定义）全部是接口，而对于成员变量全部是实现了，所以为了将接口和实现中分离，可以采取将类中原来所有的数据都放在一个impl类中，而当前类的成员数据替换成impl类的一个指针 ；然后对于在类的声明式中用到了其他的类，使用类的前置声明即可，如class Date;