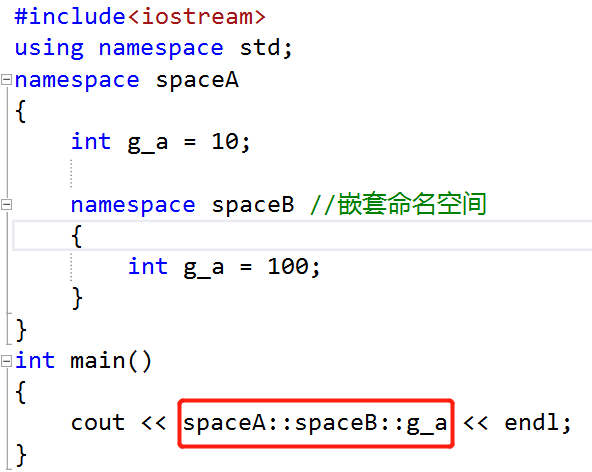
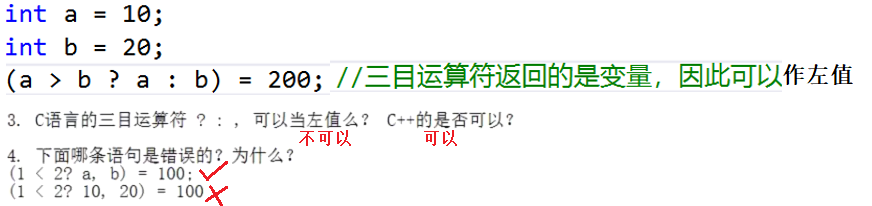
# ==========C++增强============

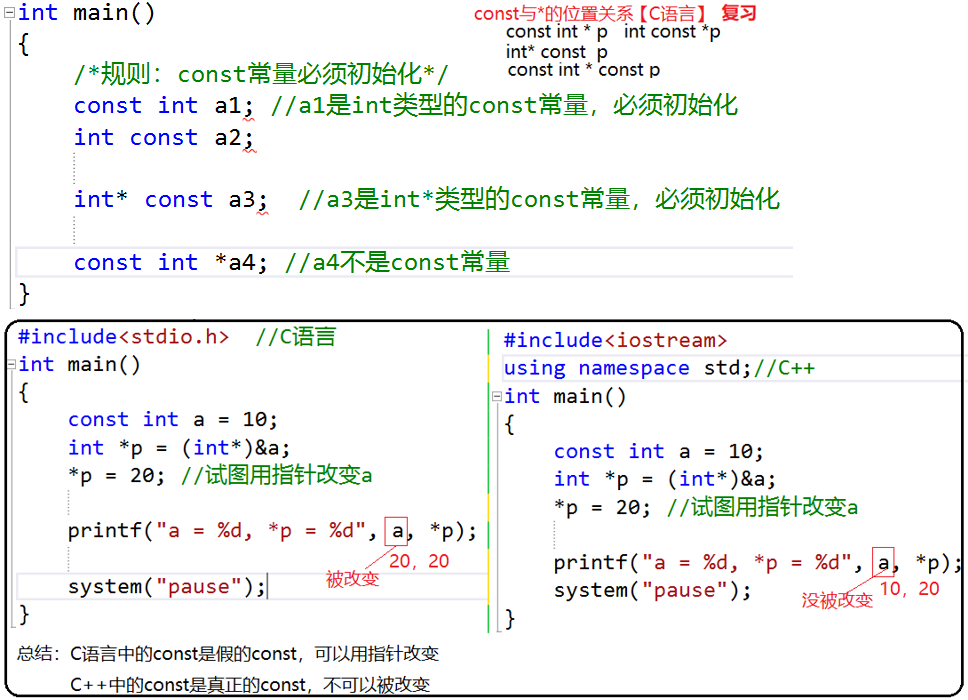
# 1.命名空间namespace



# 2.三目运算符：可以当左值



# 3.const：常量区

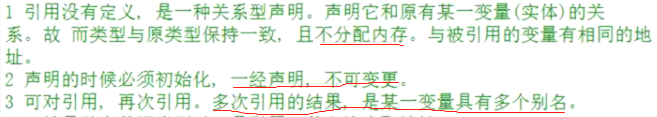


### const修饰引用 （const引用，最常作为函数形参）

# 4.枚举类型enum

枚举类型的变量，只能赋值枚举类型

# 5.引用（变量的别名）



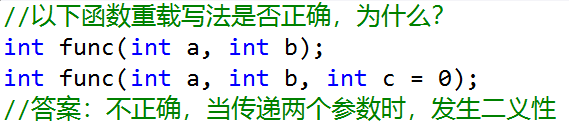
## 注意：引用作为函数的返回值



# 6.内联函数inline 与 宏定义#define



# 7.函数重载

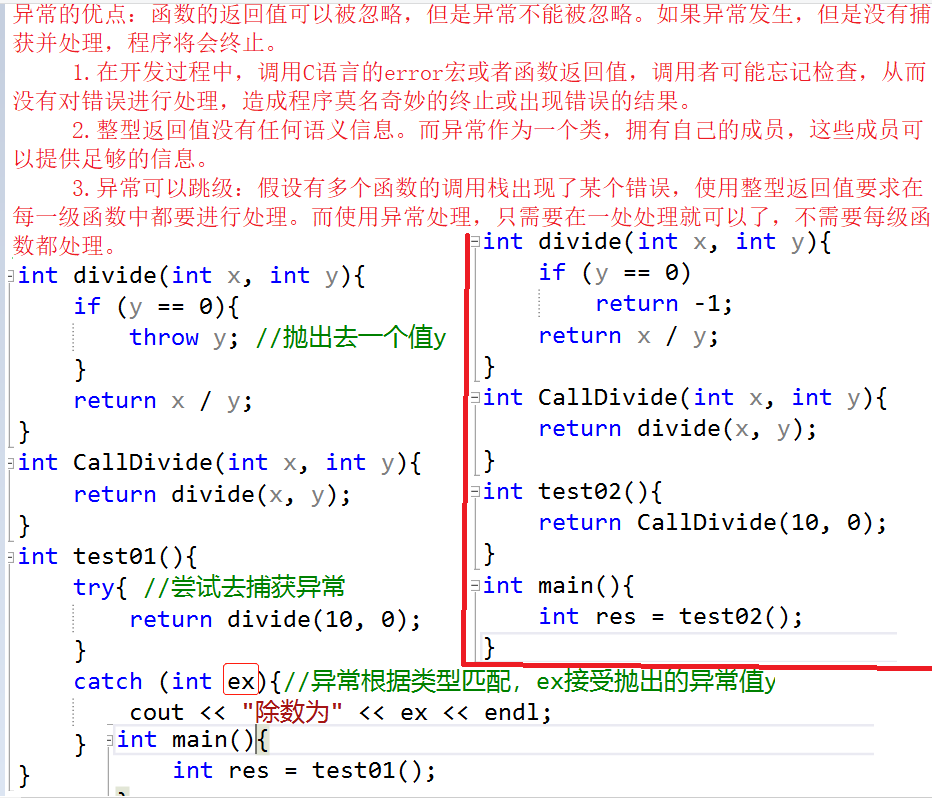


# 8.四种类型转换

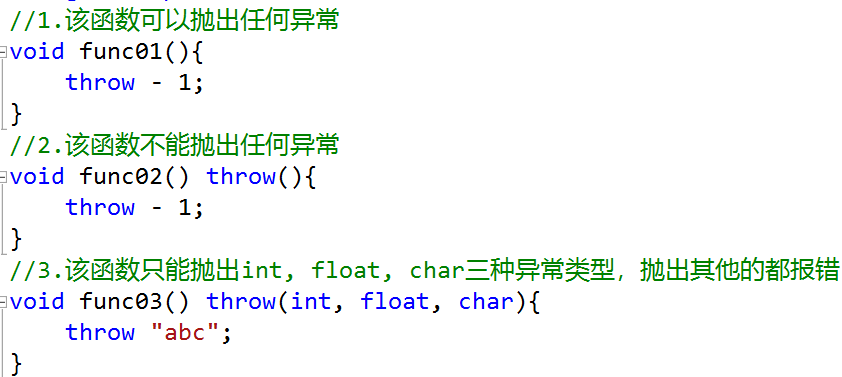


# =========异常处理========

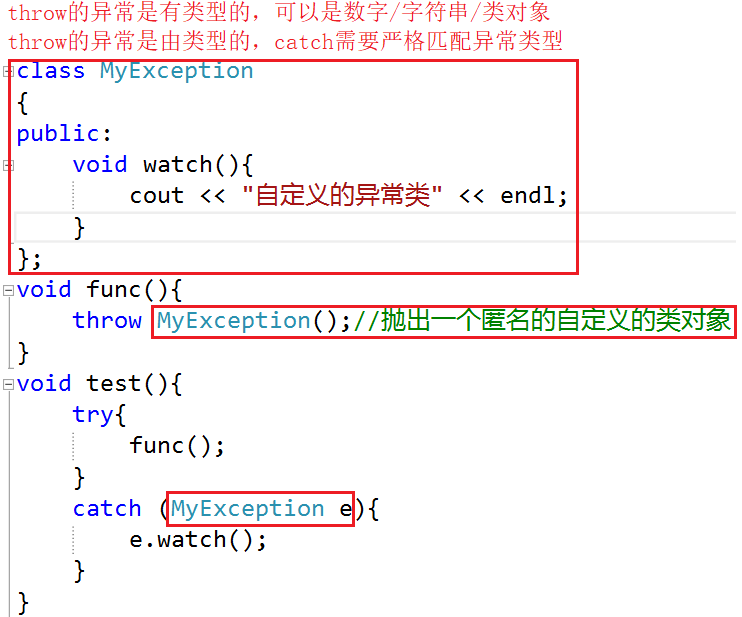
# 1.异常优点和使用



# 2.异常函数的声明



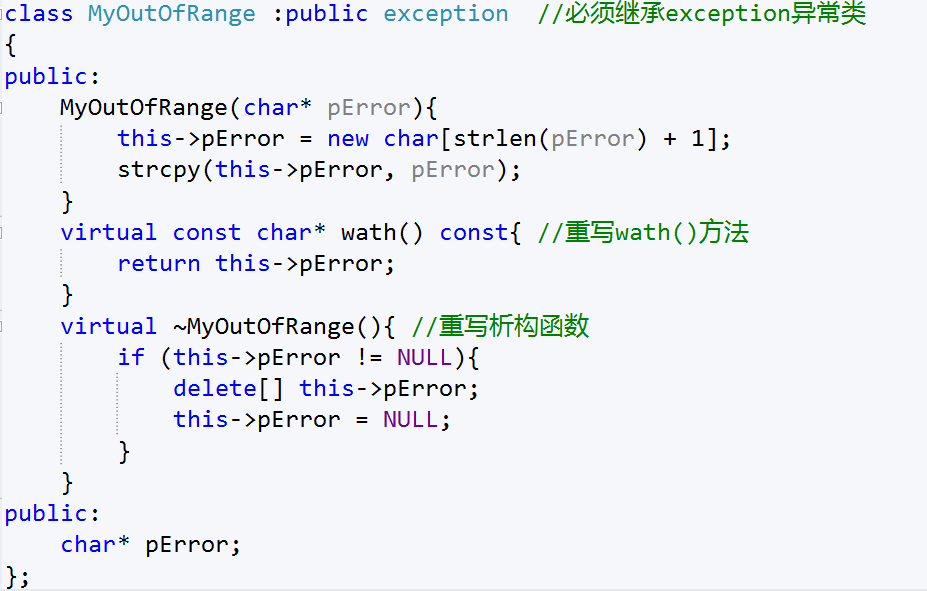
# 3.异常类型与自定义异常处理类



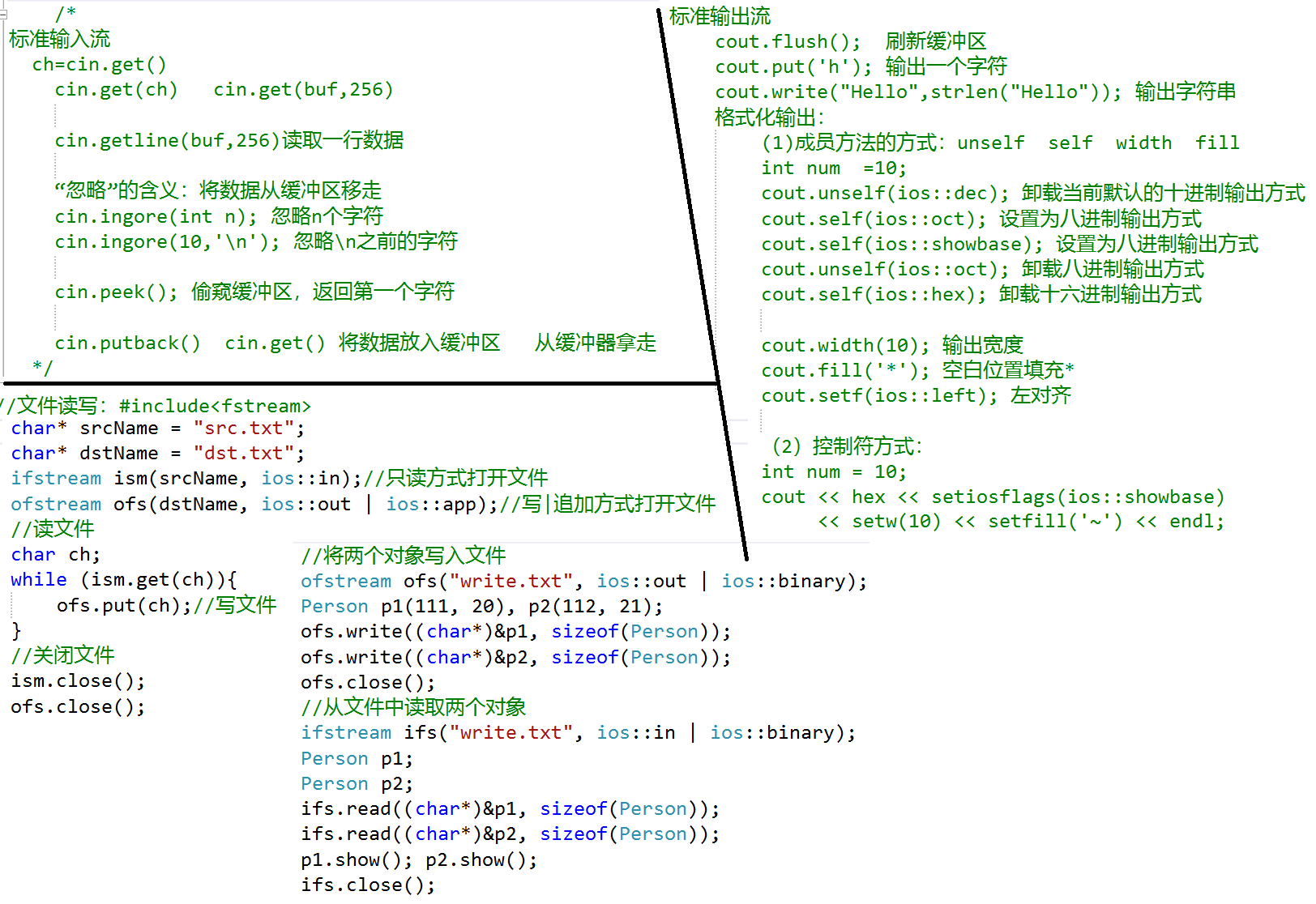
# 4.异常对象的生命周期（异常的对象/引用/指针）



# 5.编写自己的异常类（必须重写Exception类的what函数和虚析构函数）

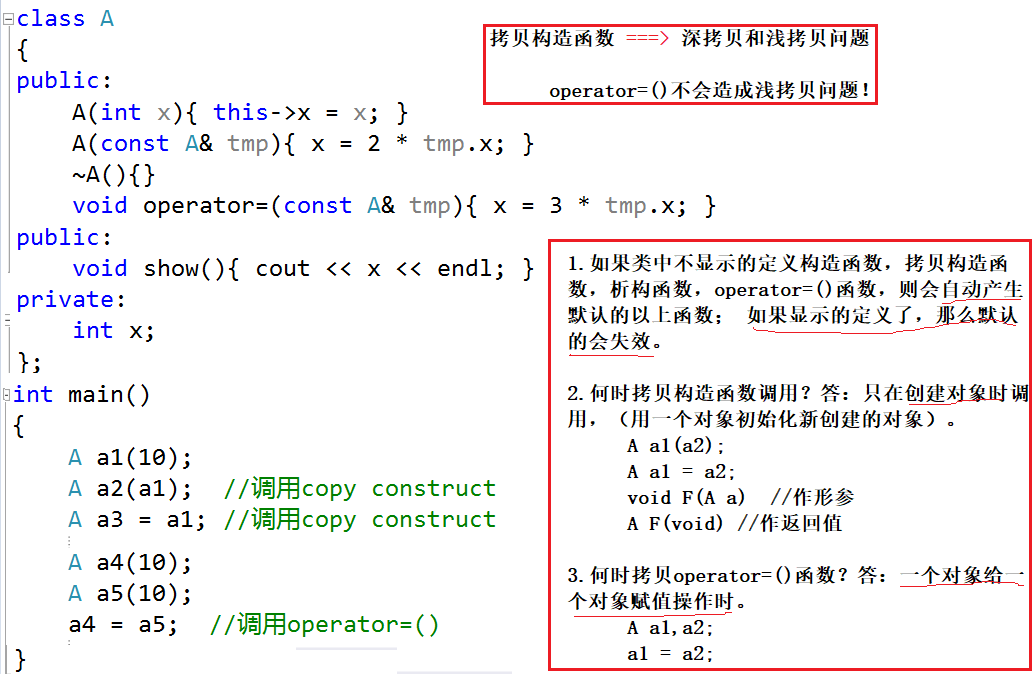


# =========C++输入输出流========

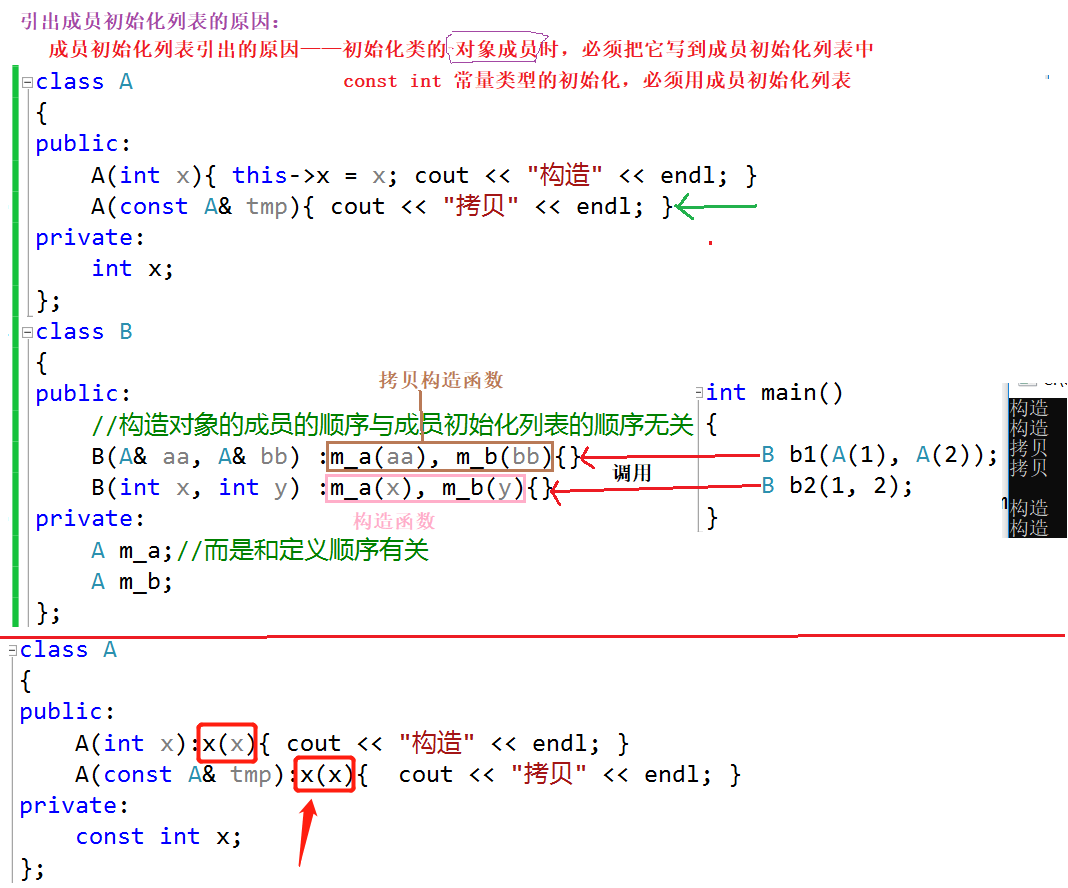


# =========类和对象==========

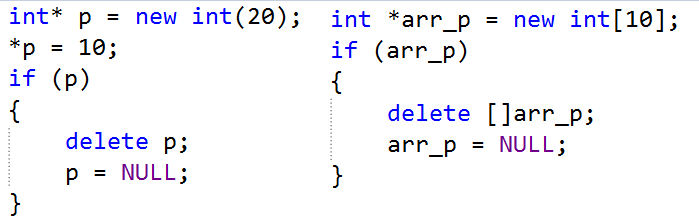
# 构造函数/析构函数/拷贝构造函数/operator=()



2.**成员初始化列表**



# 2.new/delete 运算符

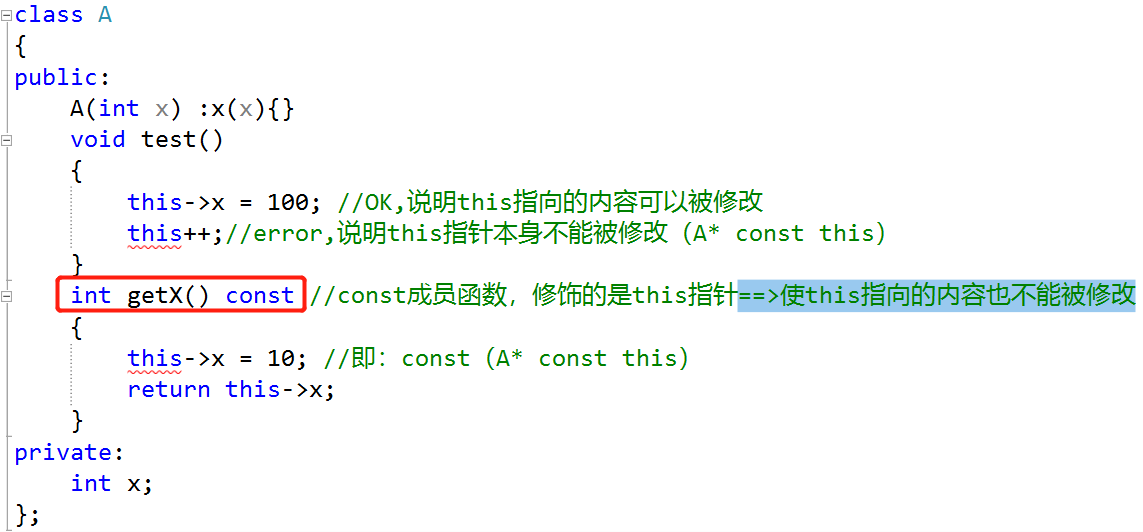


# 3.静态成员变量和静态成员函数

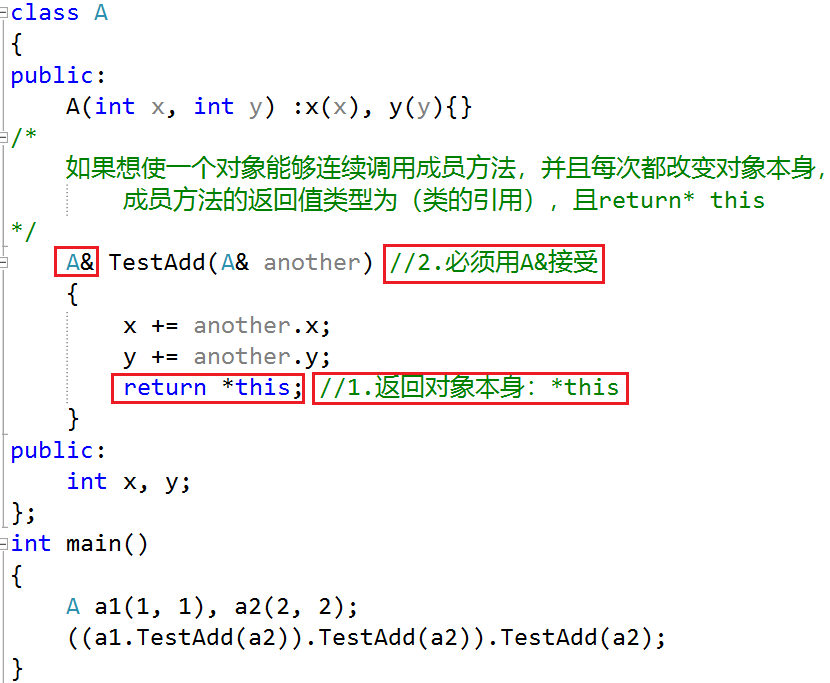


# 4.this指针（区分变量属于哪个对象）

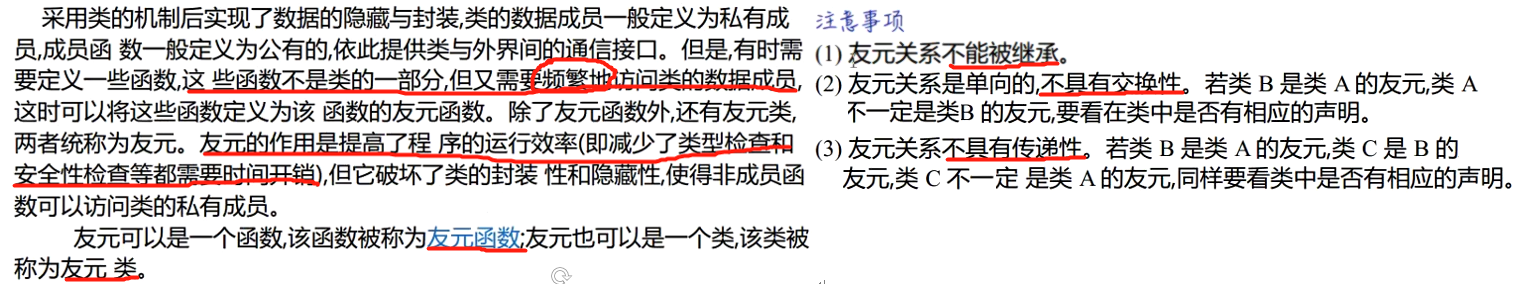
## （1）this的本质 const修饰成员函数

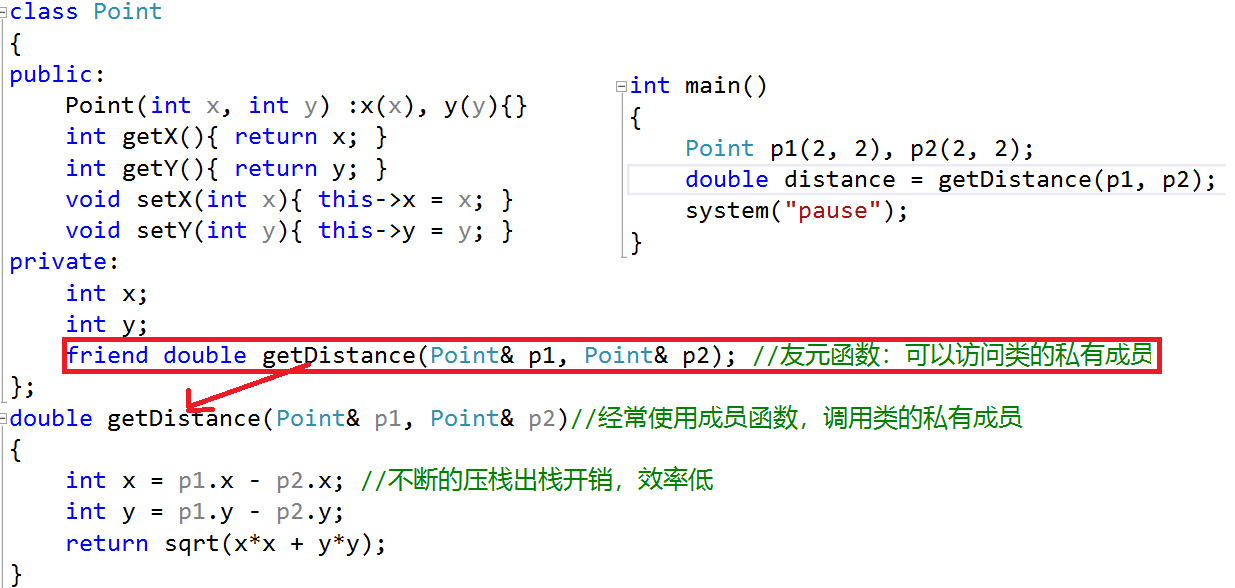


## （2）返回对象本身



# 5.友元：友元函数/友元类

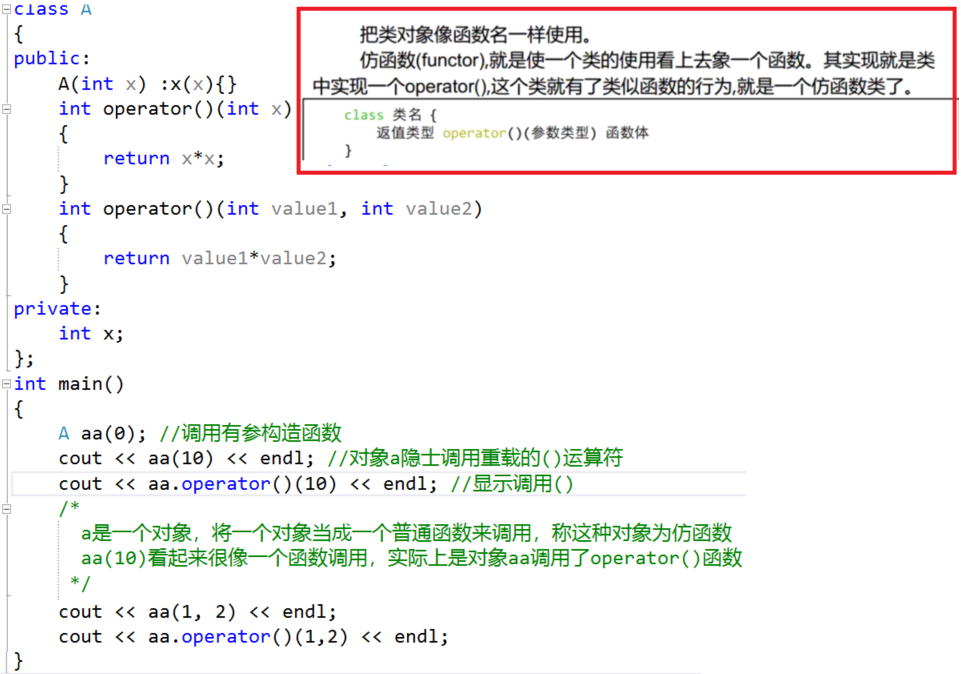




# 6.操作符重载（类的成员函数，类的友元函数，类外的普通函数）



# 7.operator() 小括号运算符重载



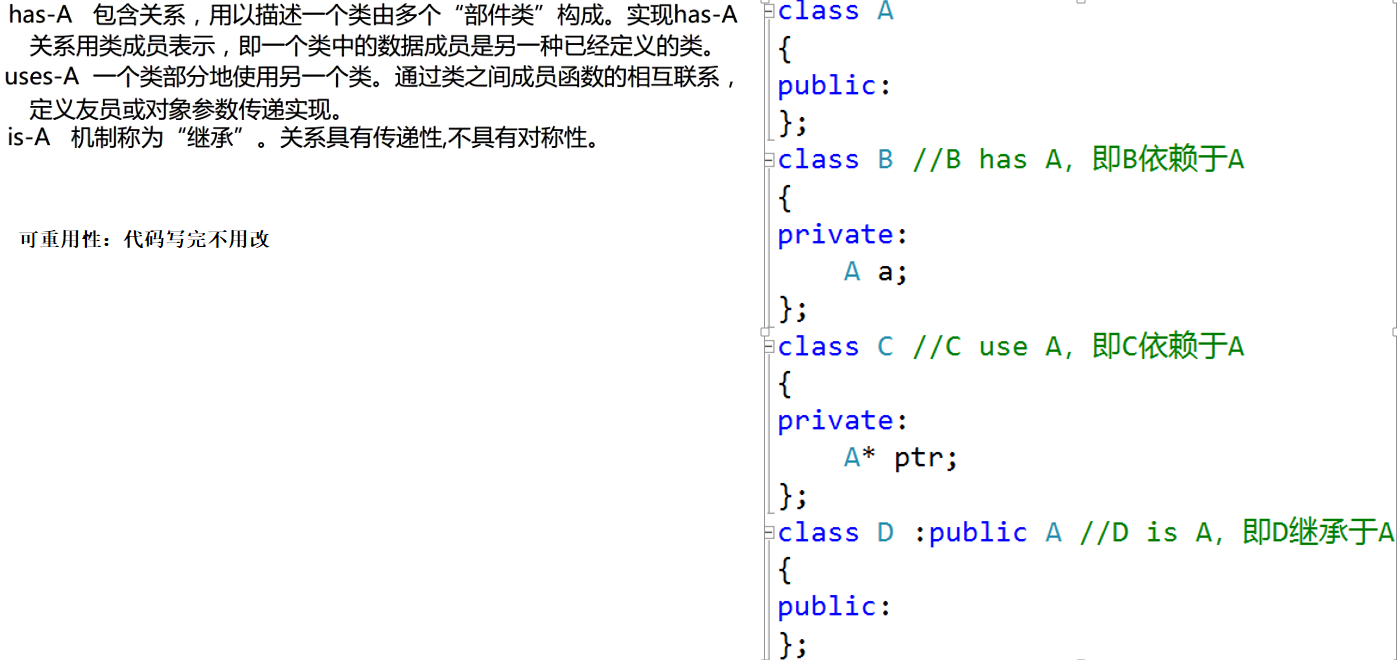
# 8.operator new和delete

# 9.自定义智能指针



# 10.综合练习：自己实现string类（详细见github）

# =======类和类之间的关系=======



# 1.继承



# 2.类的赋值兼容问题（多态发生的必要条件）

子类对象可以赋值给父类对象；子类指针可以赋值给父类指针

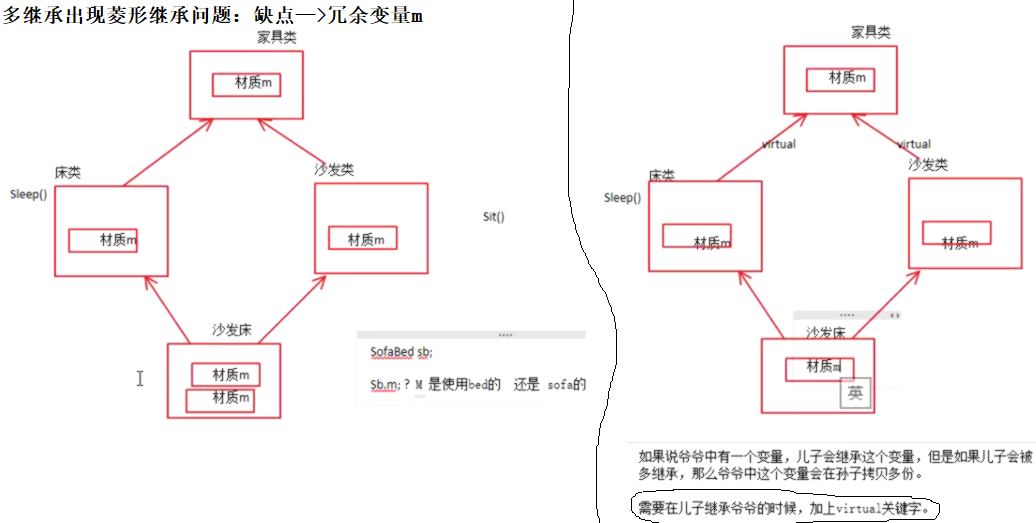
但是反之，都不可以。

# 3.子类父类出现重名函数和变量

如何访问？——父类::

# 4.static成员变量（子类父类都共用同一个）

# 5.多继承与虚继承



# 6.多态（面向对象的核心）

多态发生的三个必要条件：继承+虚函数重写+父类指针或引用指向子类对象

# 7.静态联编/动态联编

静态联编：程序的匹配和连接在编译阶段实现，也叫早绑定（重载函数使用静态联编）

动态联编：程序联编推迟到运行时实现，也叫迟绑定（switch…if…就是动态联编）

多态对于编译器来讲，是动态联编（迟绑定）

# 8.虚析构函数（为什么析构函数定义为virtual）

防止父类指针指向子类对象时，用delete指针时，不能释放子类内存，导致内存泄漏

A\* ptrB = new B(10);

delete ptrB;

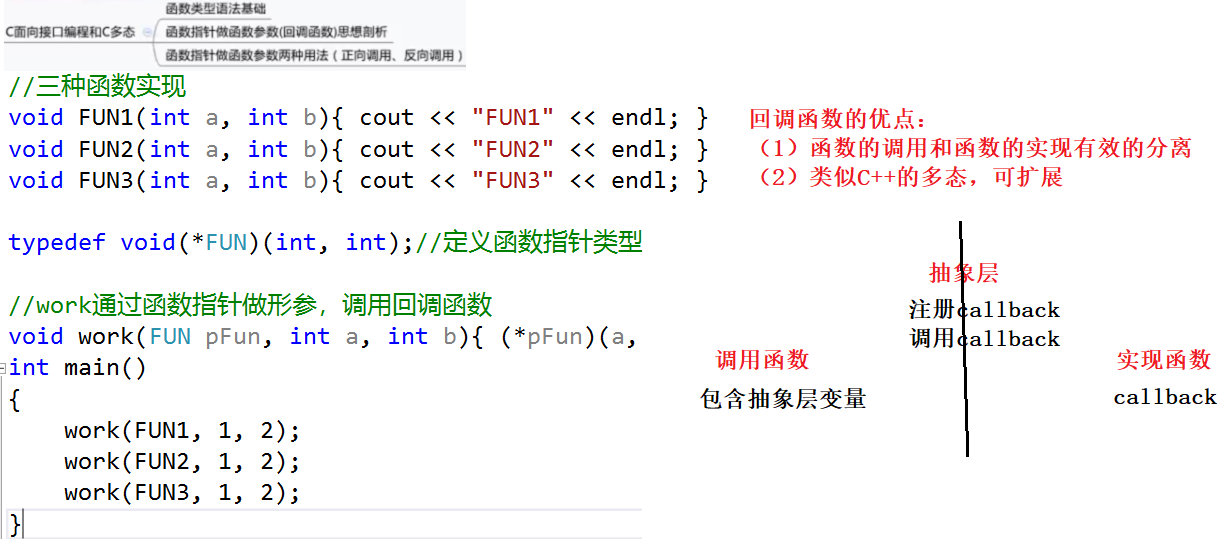
# 9.纯虚函数和抽象类≈≈≈≈≈Java中的接口

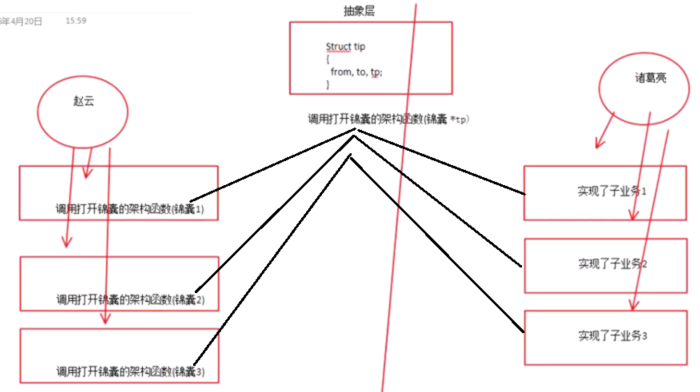
（1）具有纯虚函数的类，叫做抽象类

（2）抽象类不能被实例化

（3）继承抽象类的类，如果没重写纯虚函数，那么该类也是抽象类，不能被实例化

# 10.\*\*CallBack函数指针实现多态\*\* 赵云诸葛亮（锦囊妙计）





# ======多态面试题======

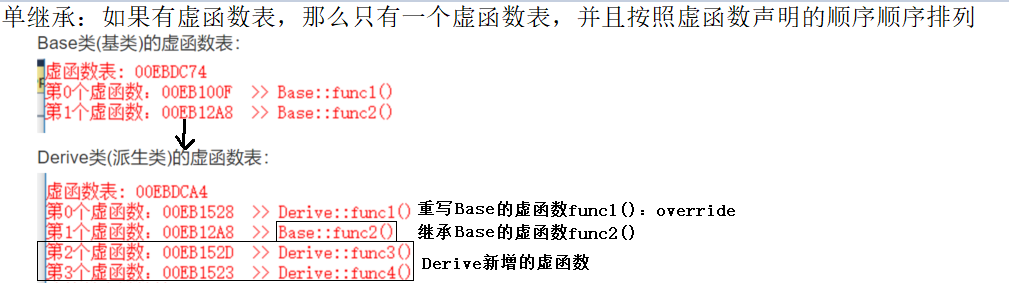
# 1. 多态的实现原理（虚函数表）

# 2.面试题：vptr指针的分步初始化（构造函数中的虚函数，不能实现多态）

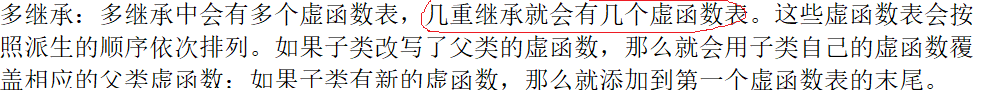


# 3.面试题：画出类的虚表VTable内容

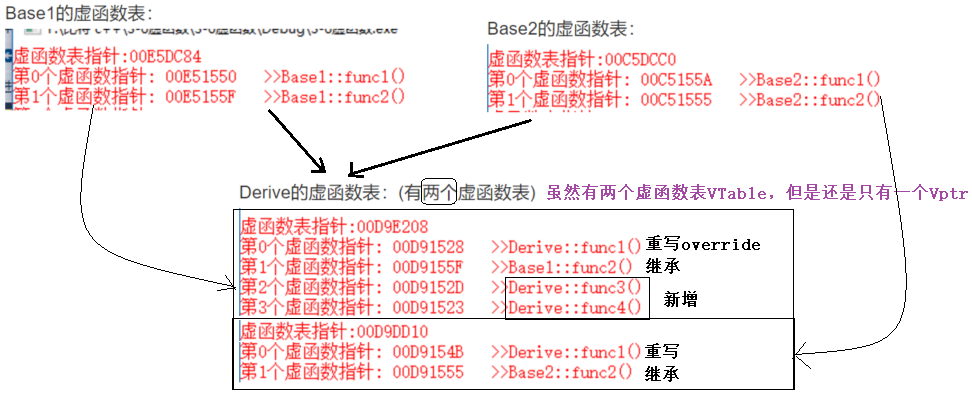
## 3.1单继承



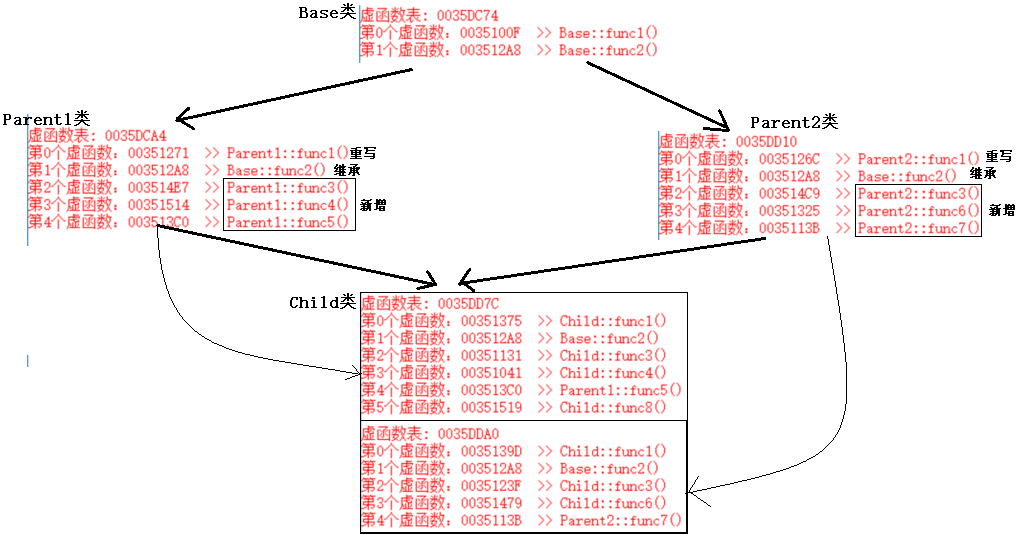
## 3.2多继承



### 3.2.1简单的多继承

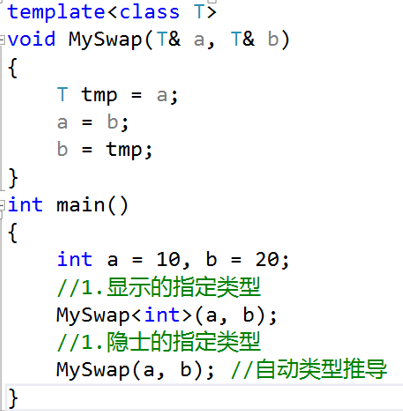


### 3.2.1复杂的多继承（菱形继承）

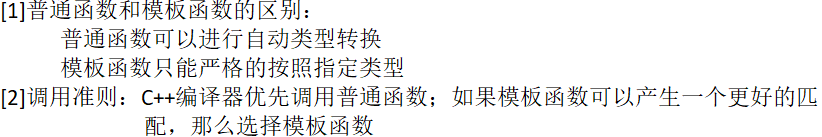


# ==============模板==============

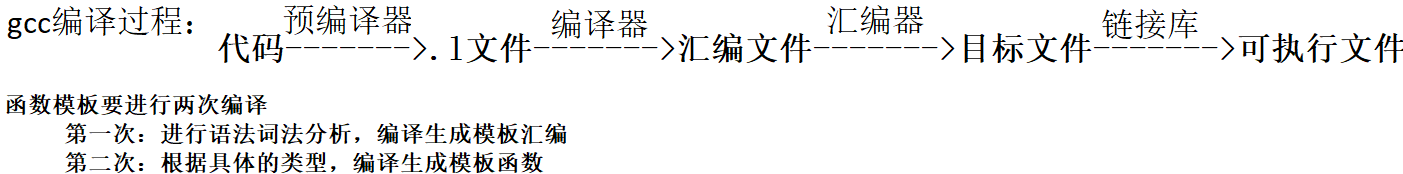
# 1. 语法：函数模板—>参数类型化：传什么类型，就执行什么类型



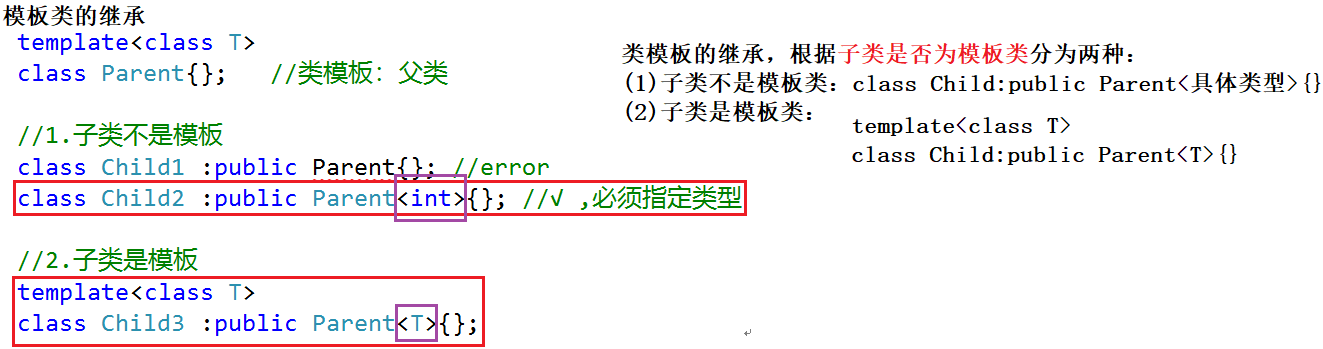
# 2.函数模板调用准则



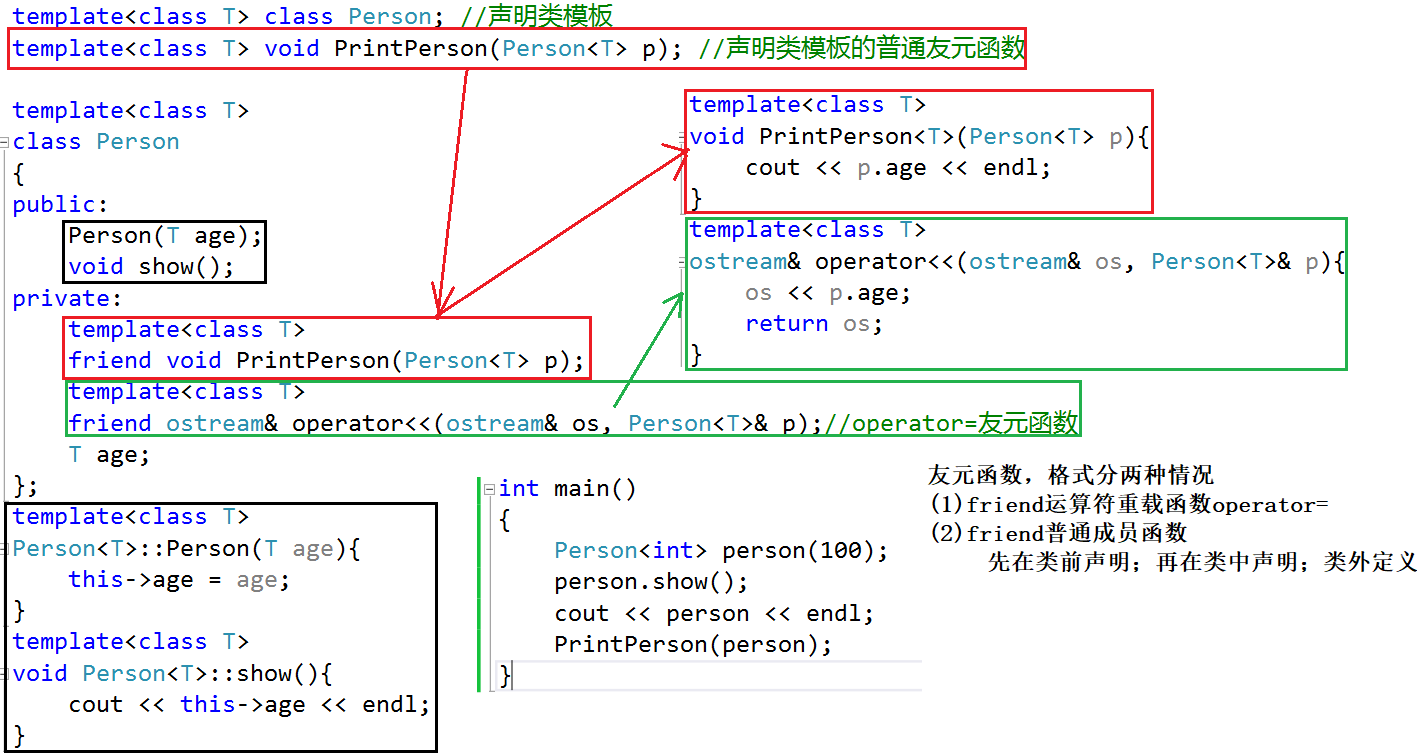
# 3.C++模板机制如何实现的？



# 4. 语法：类模板的继承

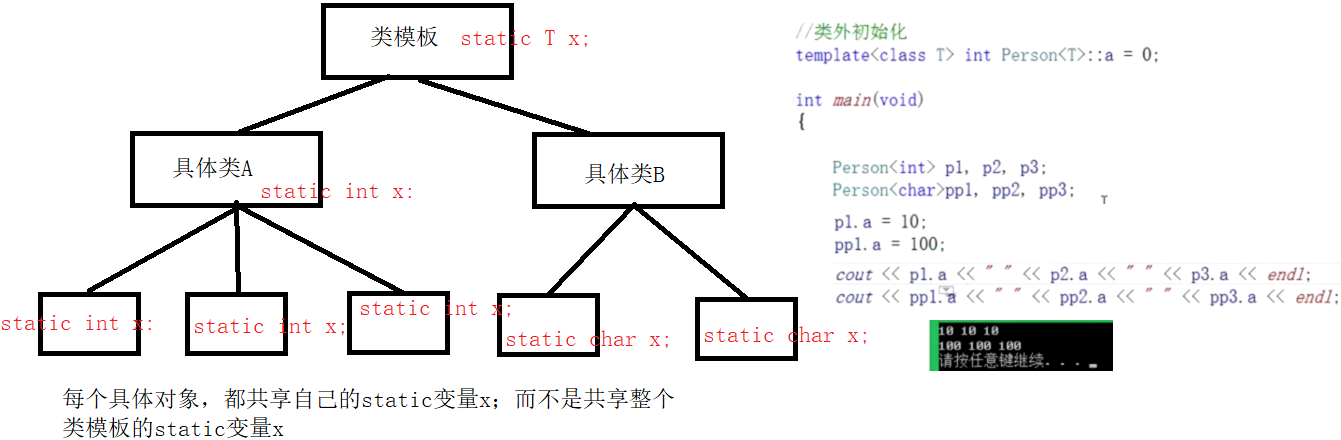


# 5.语法：类模板成员函数的类外定义//类模板的友元函数



# 6.类模板的多文件编写：写成一个.hpp，不能写成.h和.cpp

# 7.类模板与static变量



# ========STL容器========

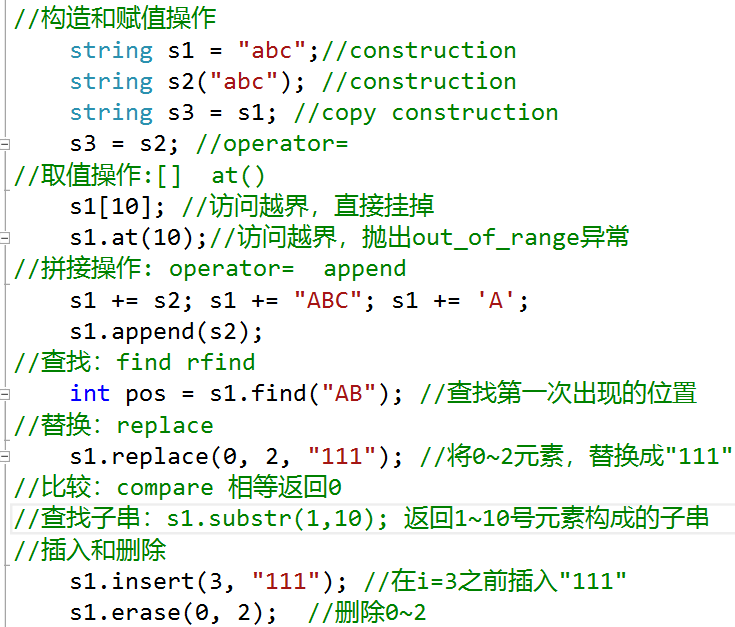
# 0.容器深拷贝和浅拷贝\*\*\* 提供的元素必须重写拷贝构造函数！

# 1.序列式容器/关联式容器

序列式容器：元素位置由进入容器个时机和地点来决定

关联式容器：元素容器有规则，进入容器的元素的位置不是由时机和地点决定

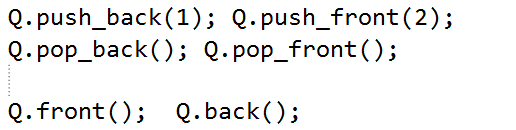
# 2.string



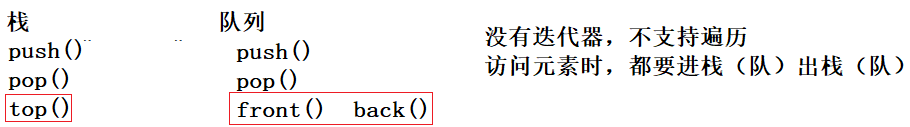
# 3.vector



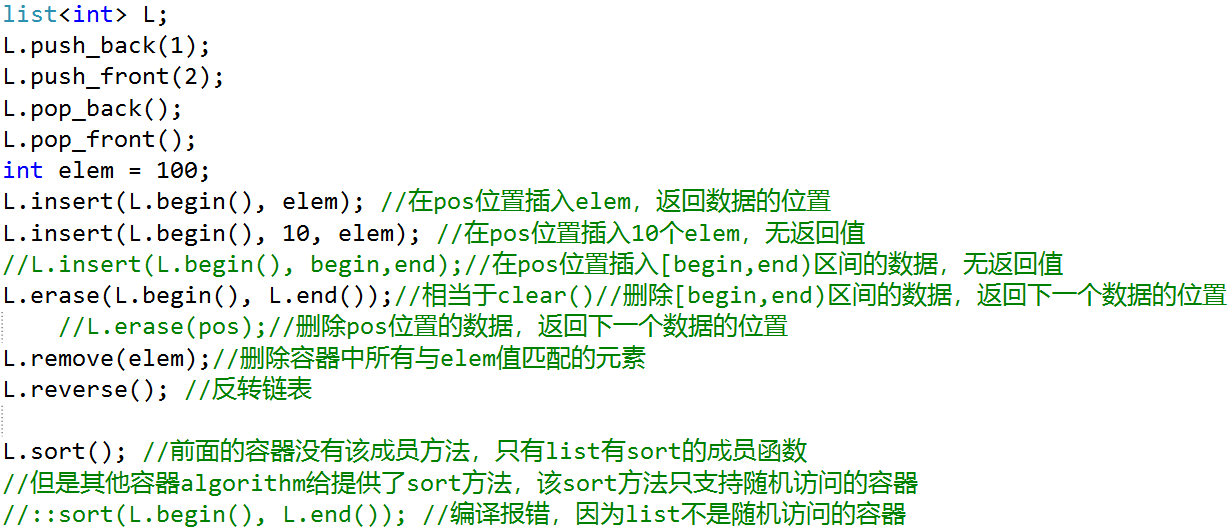
# 4.deque 双端队列（可以两端都插入/删除）：随机存取



# 5.stack和queue



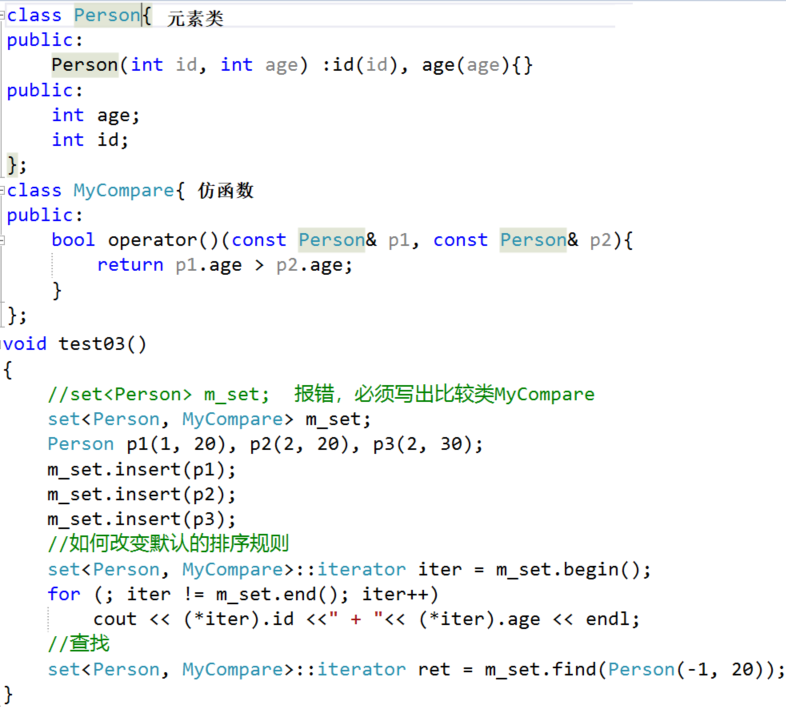
# 6.list



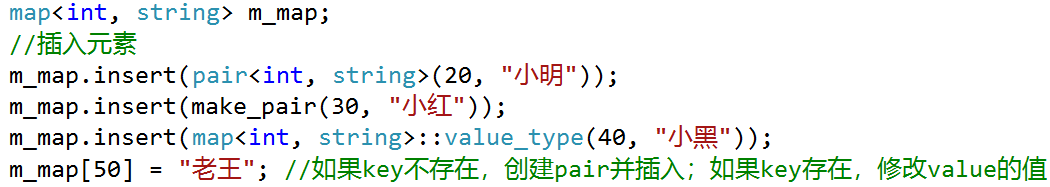
# 7.set和multiset（RBtree）



# 8.改变排序规则：必须自定义仿函数类，并重写operator()()



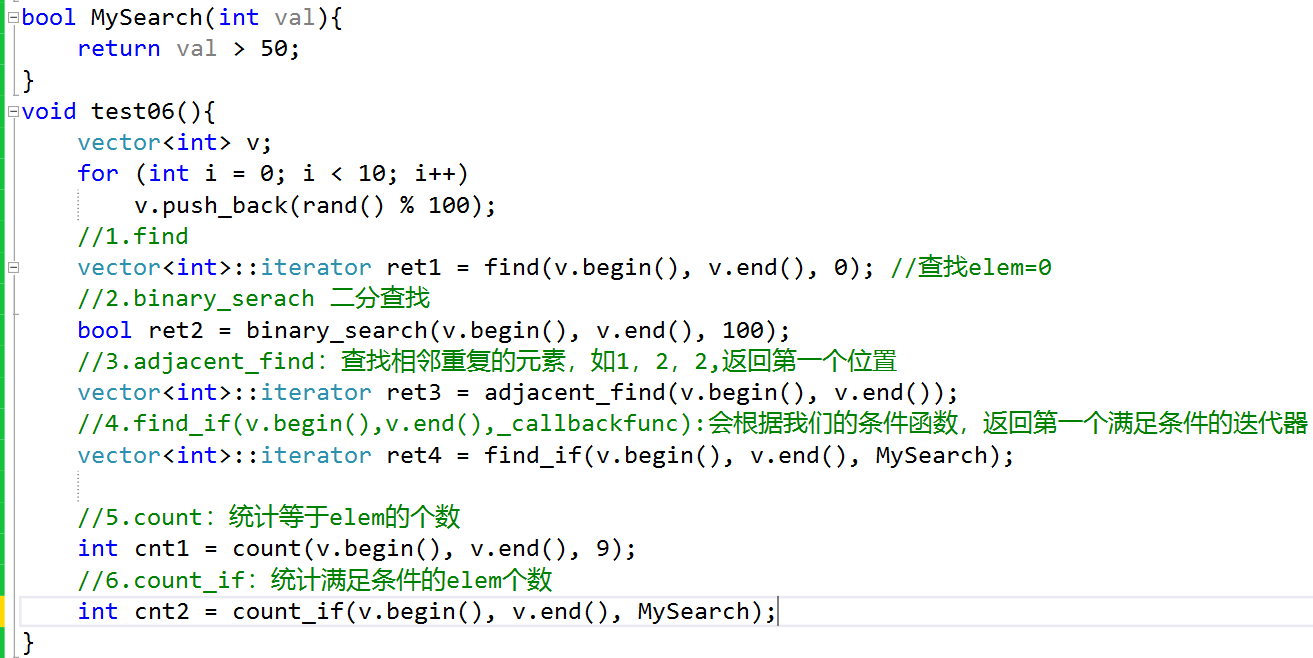
# 9.map/multimap





# ========STL算法：algorithm=======

# 1.algorithm：查找和统计

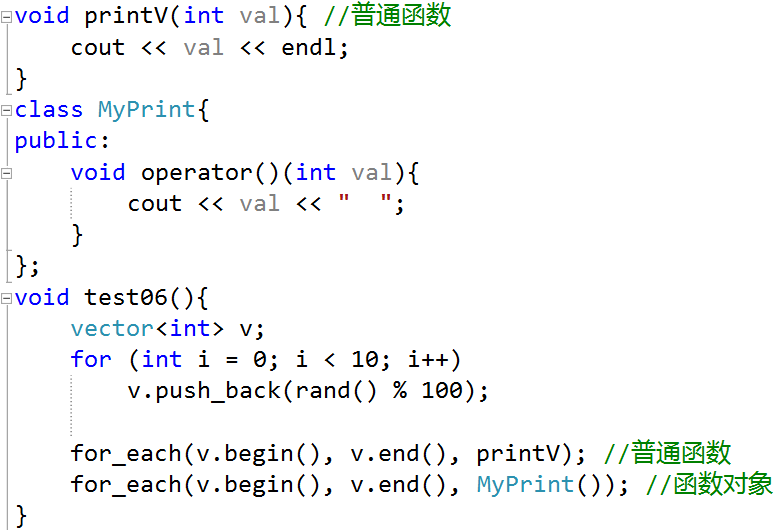


# 2.algorithm：遍历



# =======STL仿函数========

# 仿函数：必须重写operator()



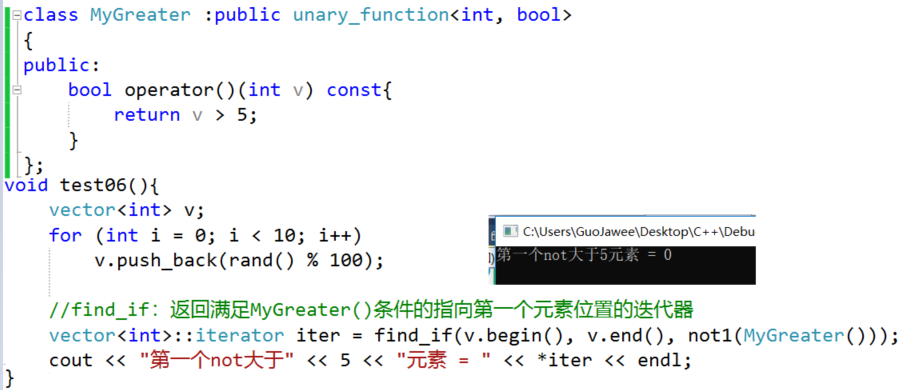
# ======STL函数对象适配器=======

# [1]绑定适配器：bind1st和bind2nd

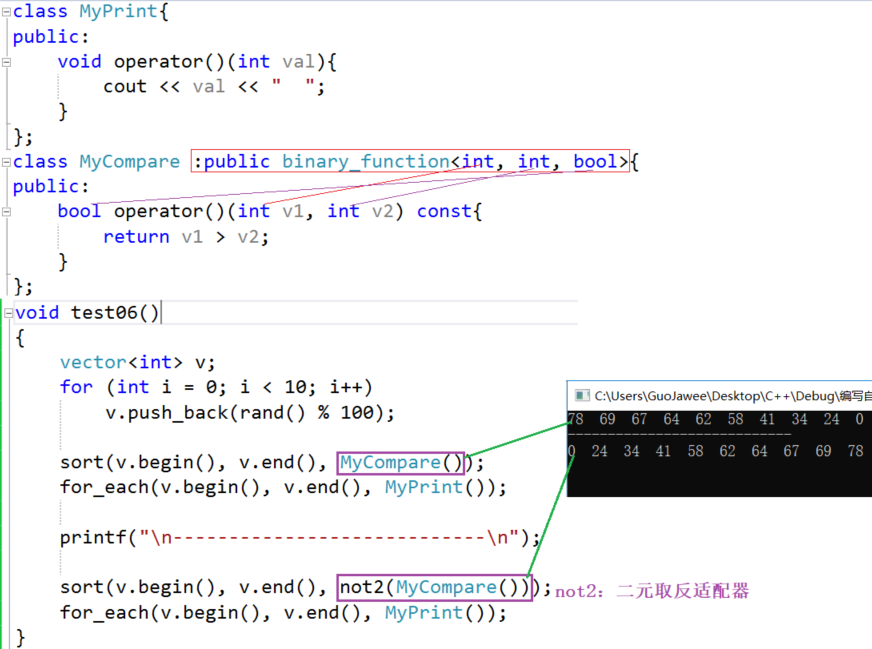




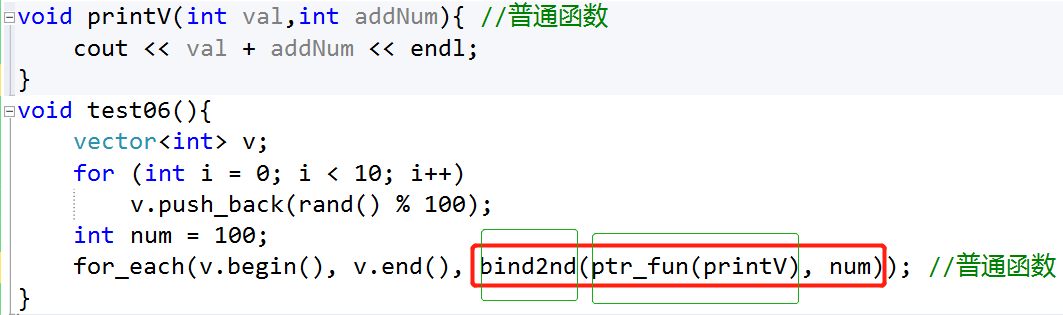
# [2]一元取反适配器：not1



# [3]二元取反适配器：not2



# [4]ptr\_fun：将普通函数转换成函数对象（使普通函数也可以适配）



# [5]成员函数适配器mem\_fun\_ref mem\_fun

