Int[n]里面的数n必须是非负值。

New int/int(1)/int[1]

Void test(int, double, int)（可以不用形参）

Void Stack<T>::push(T x) 创建对象：Stack<float> stk(N);

Int i(1)<=>int i=1

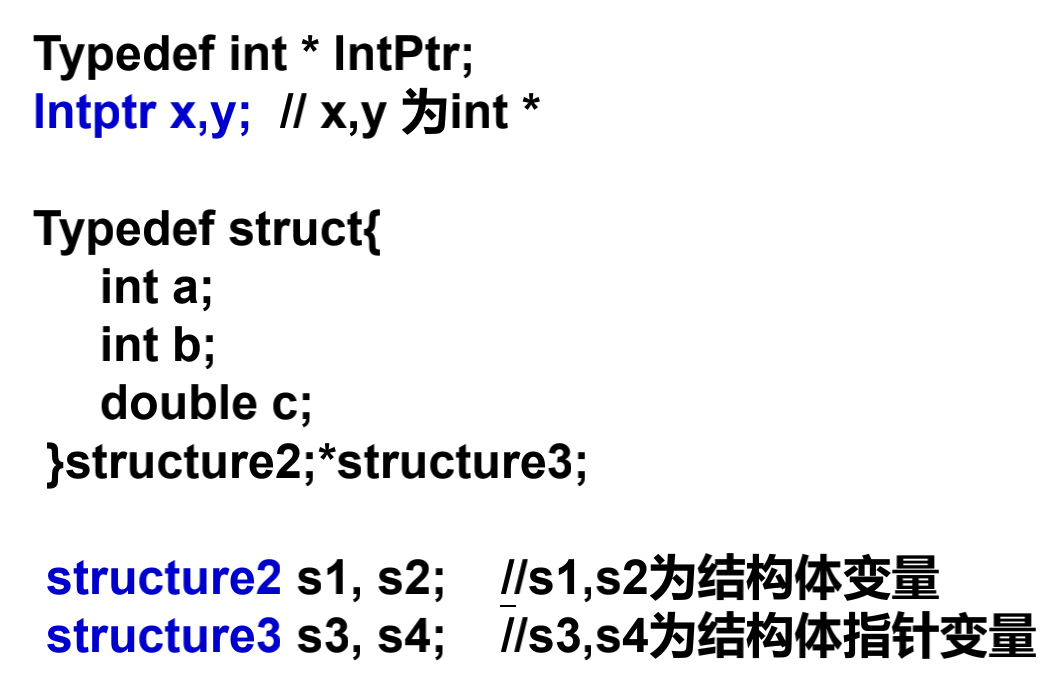
\_算字母，变量名只能字母打头，只能带字母、数字。

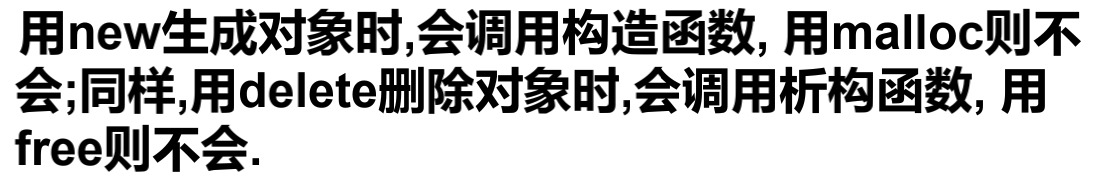
Char s[4][ ]=…错

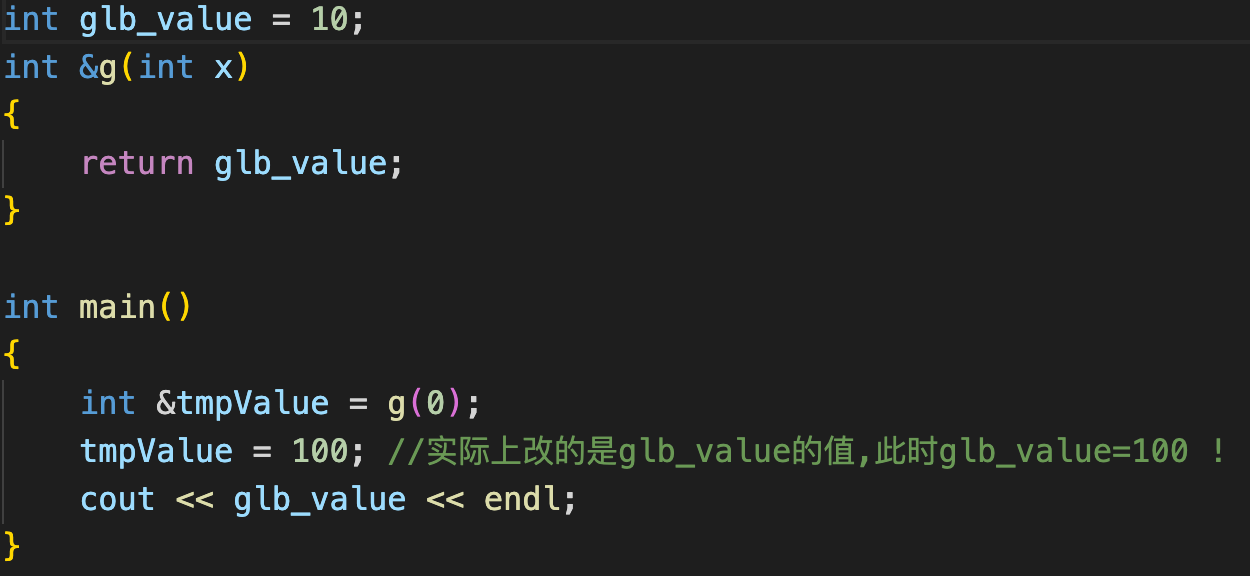
指针类型：他所指向对象的类型

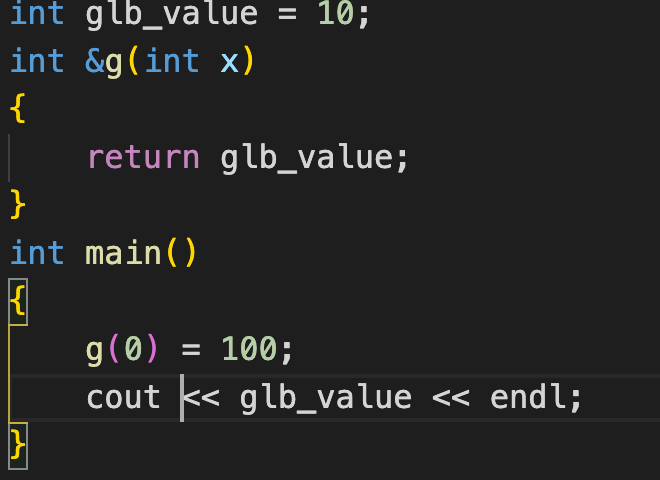
Int a[ ],a也不能修改。

引用在定义时必须初始化；初始化后，不能改变引用的 “指向”。

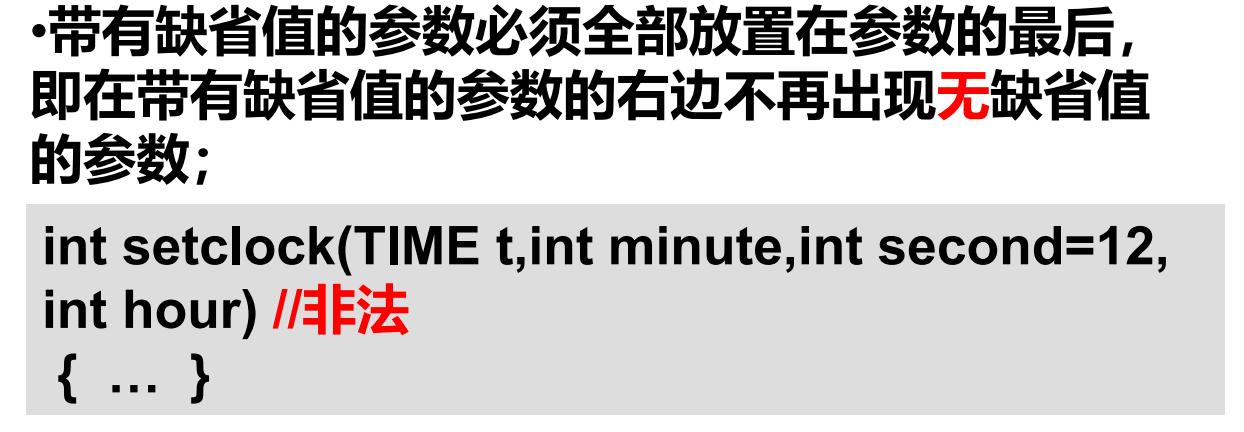


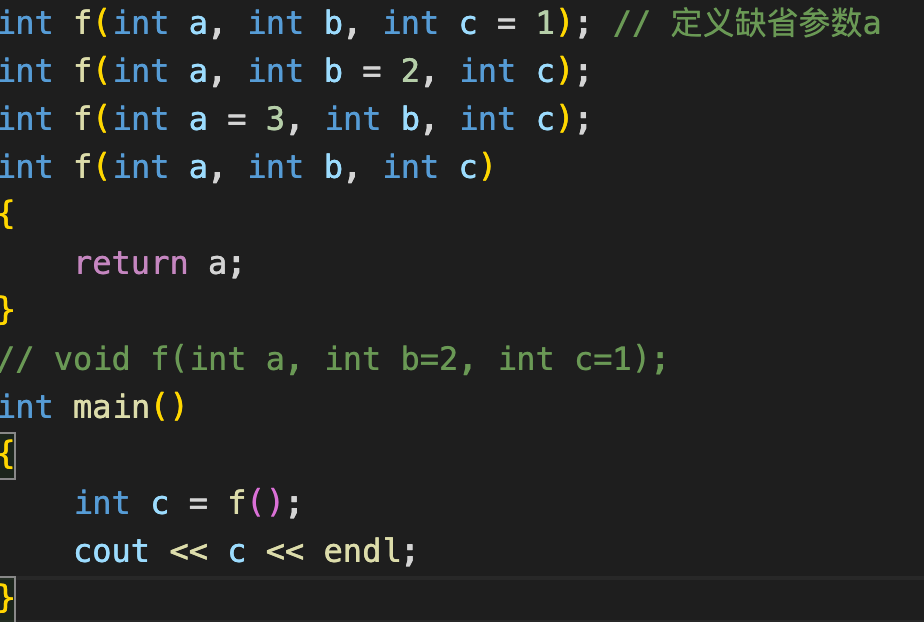




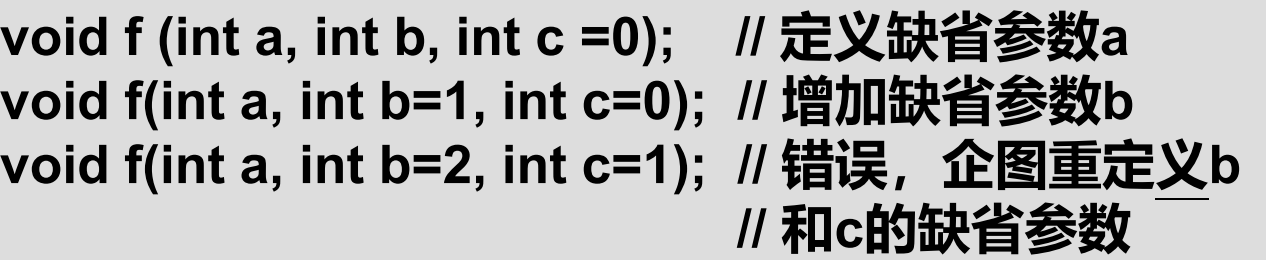


也可以直接g(0)=100

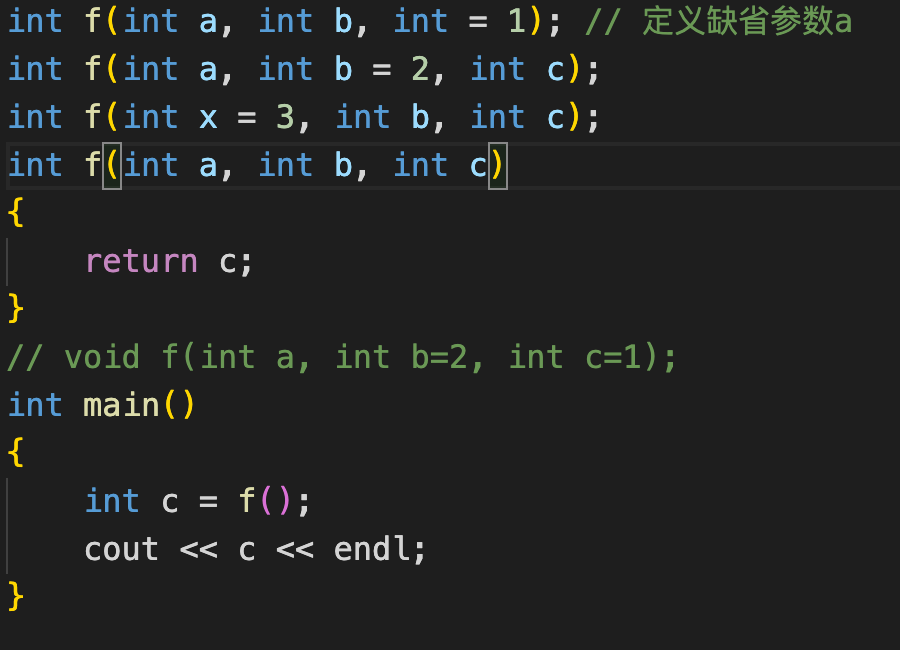




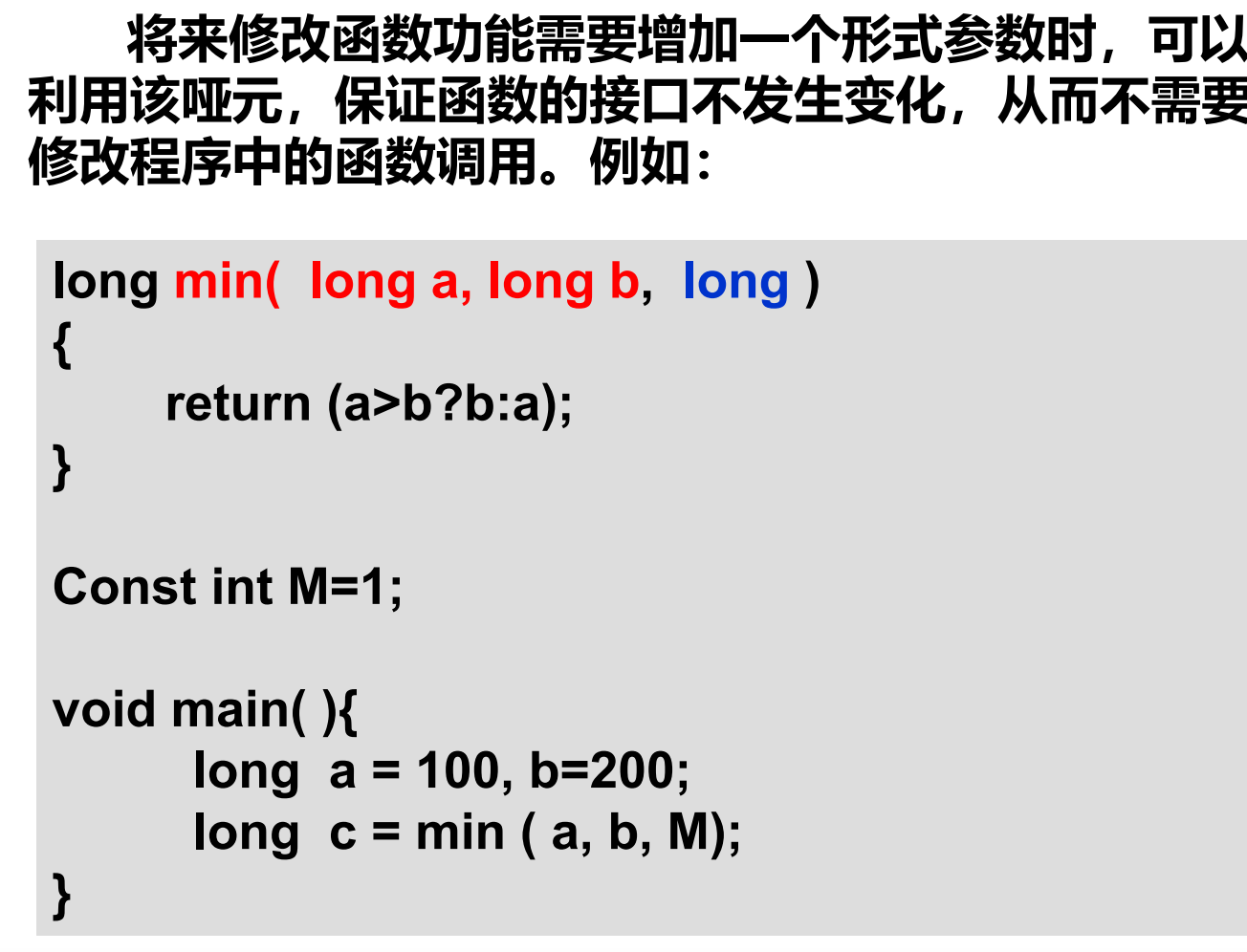
不能f(int a,int b=2,int c=1)(ppt错了)



函数形参在不同声明和定义里名字可以不同

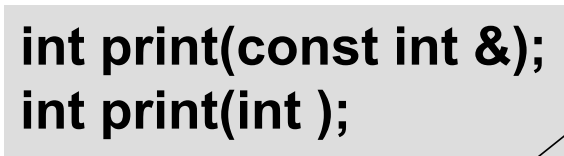


输出1



？？？？

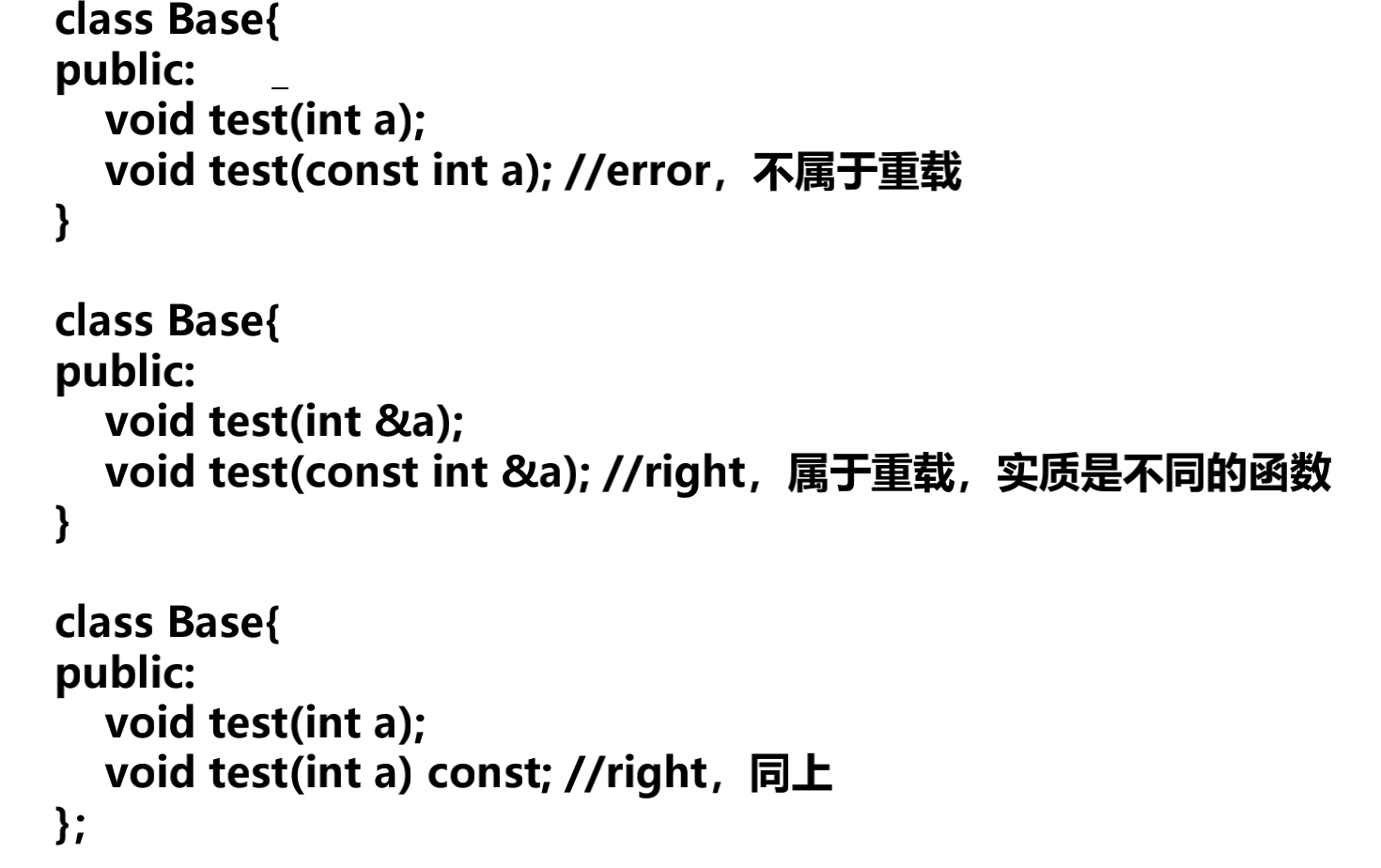
只有返回类型不同的函数不能被重载

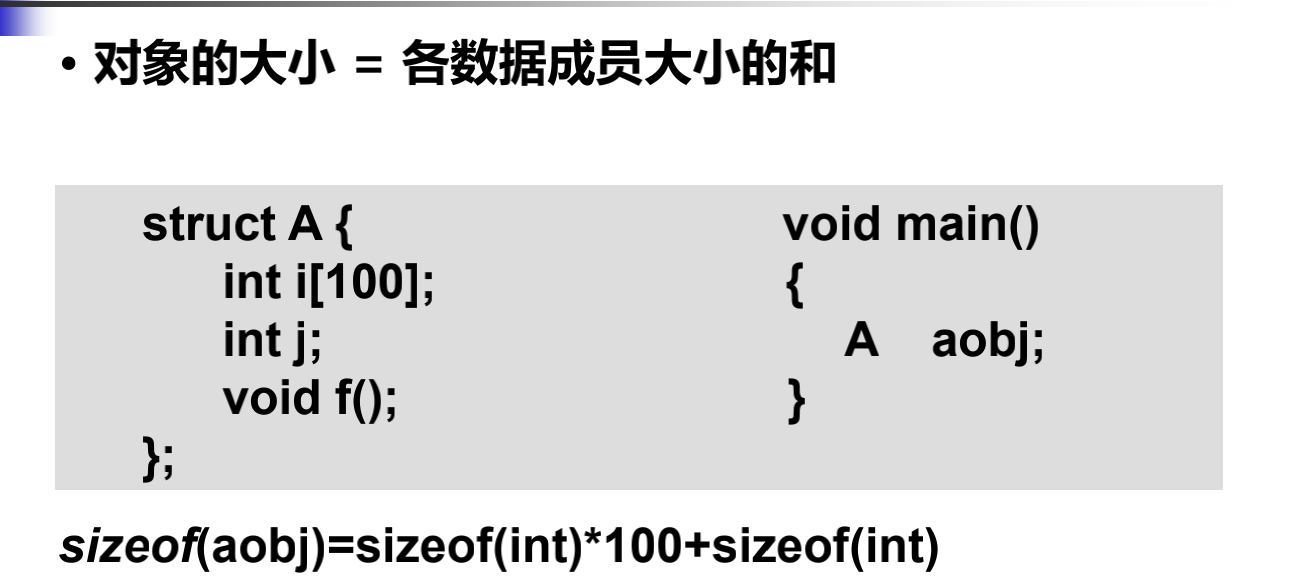


Int vs Const int, const int &, int &都不行 int& vs const int &可以，由int和const int区分

其他两种组合都不行

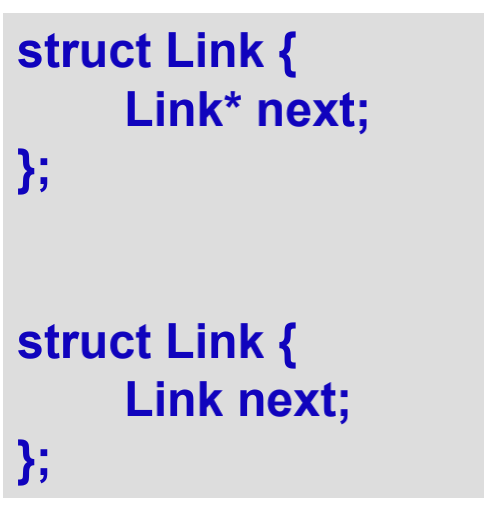
有int&和const int&时由int和const int区分，但只有const int&时int也能传





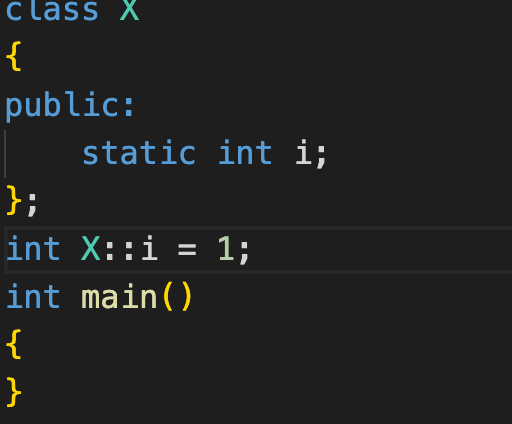
不包含函数。

没有数据成员时为1



上面对 link\*大小已经确定（指针一般为8）





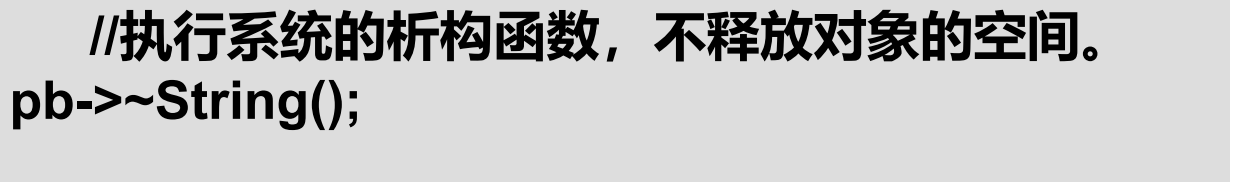
不能没有static，否则只能在里面定义

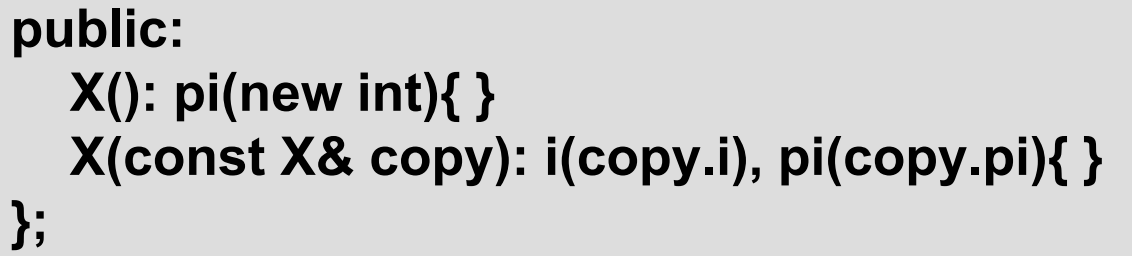
实参里带&，是取地址，形参里&是引用



main结束，不管有没有消除都会执行析构函数。

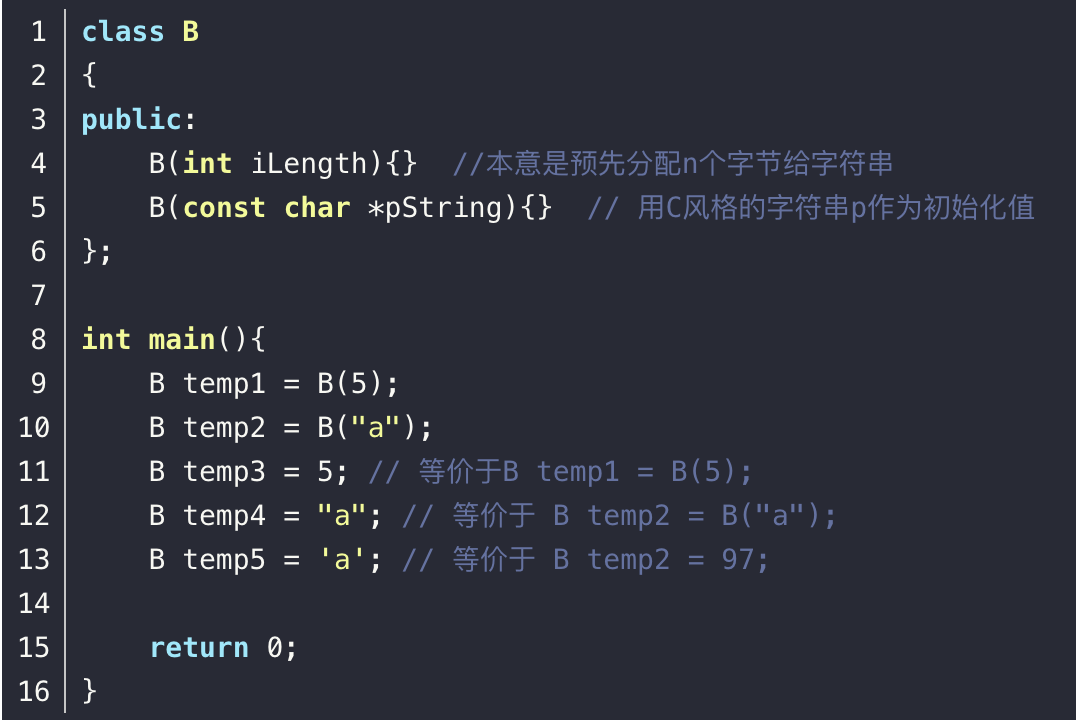


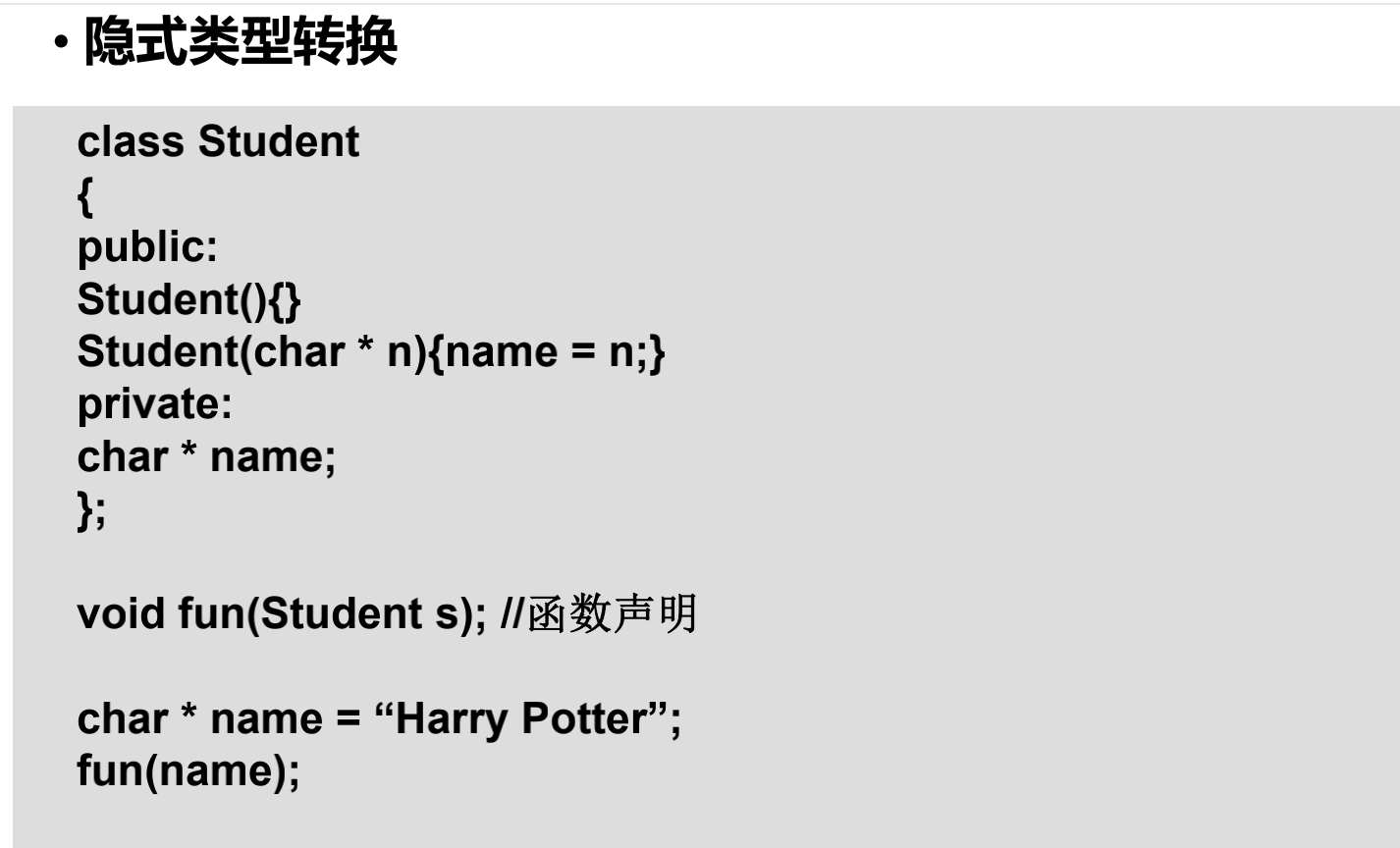


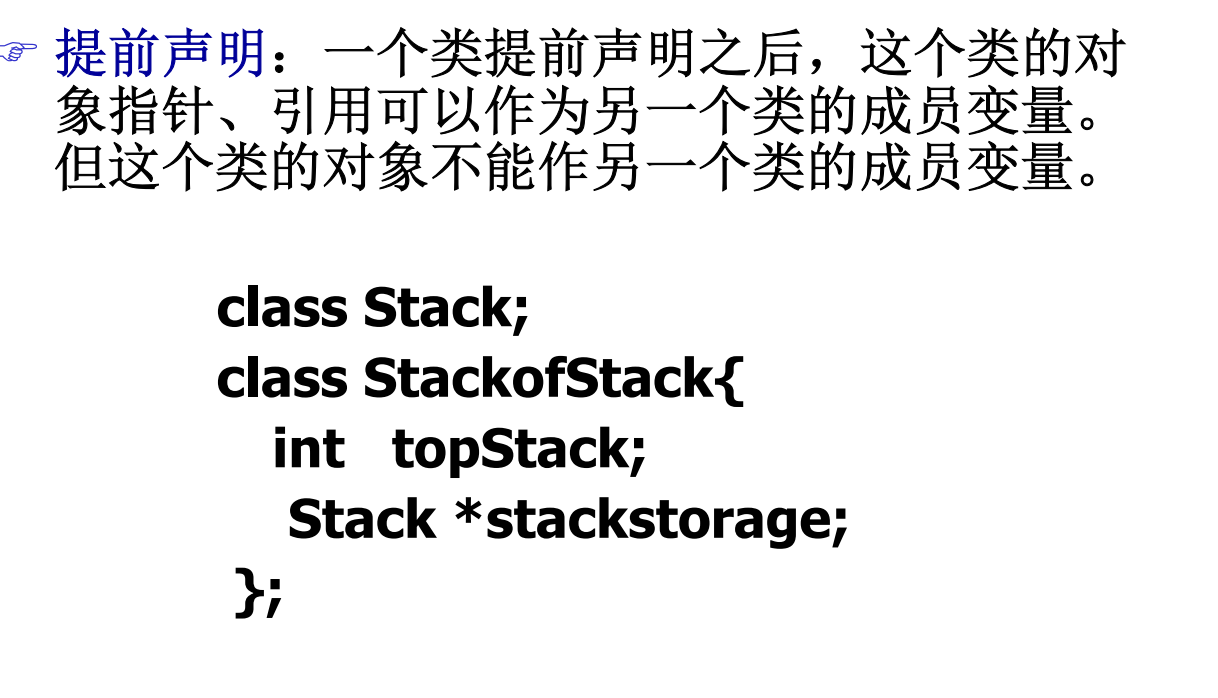


i(x)赋值 浅拷贝

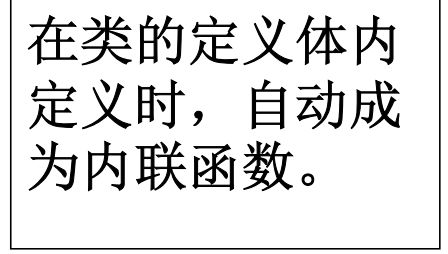




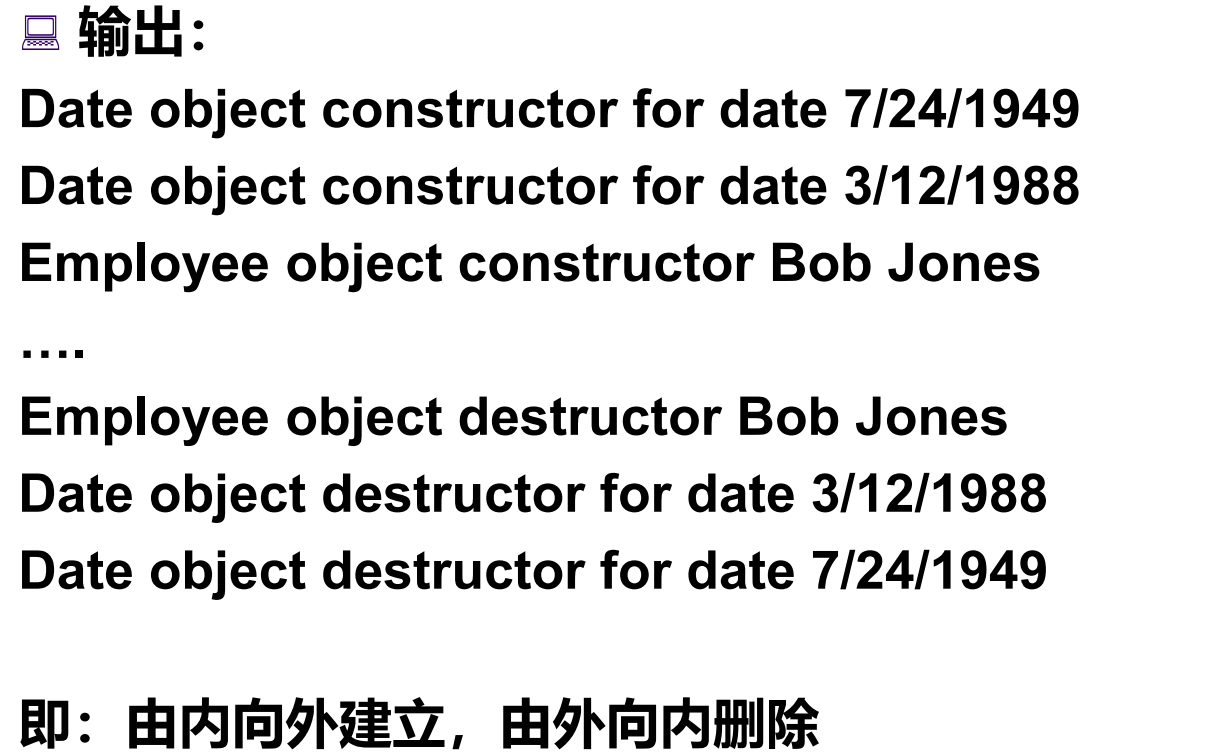


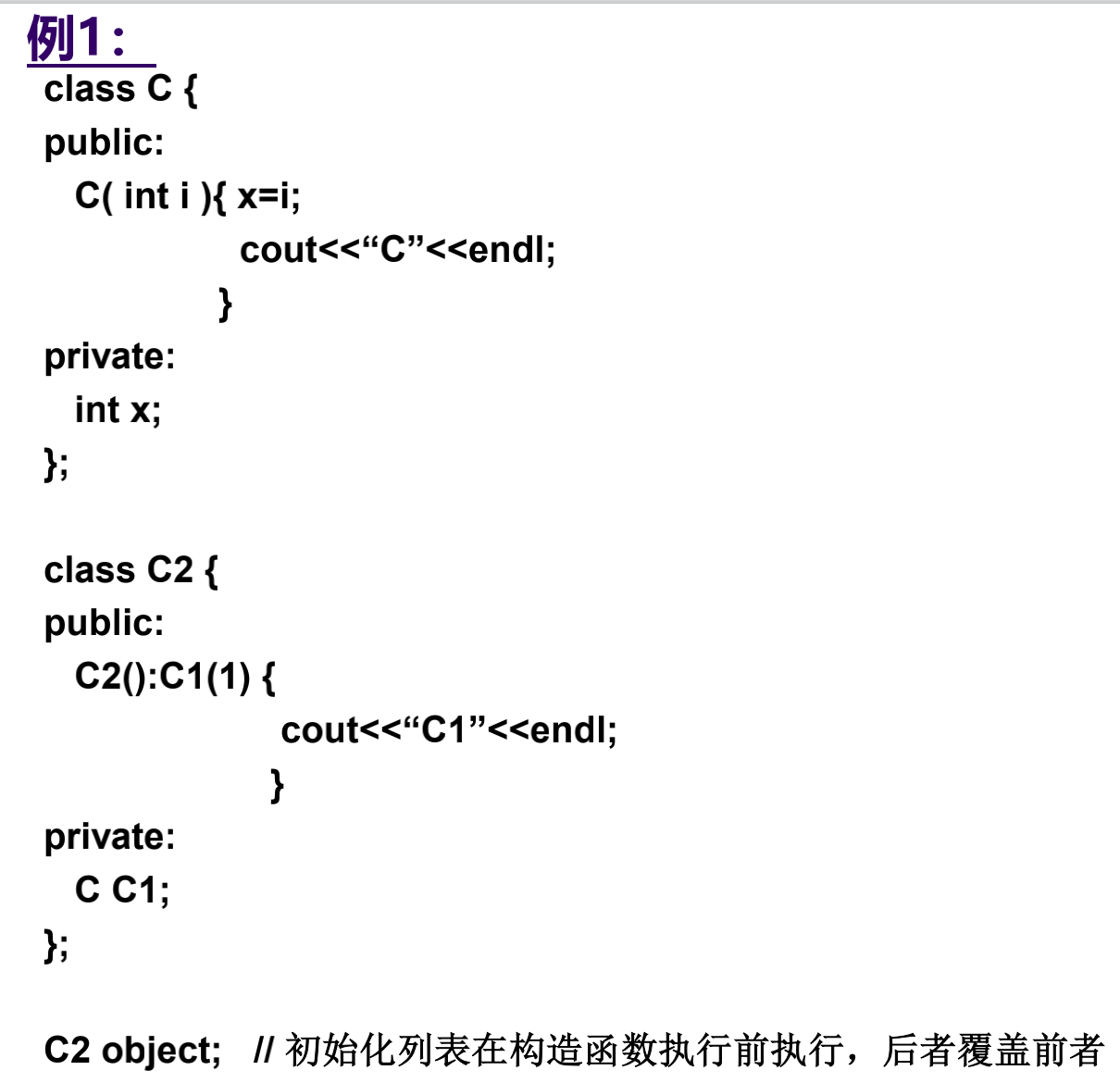


定义了就可以



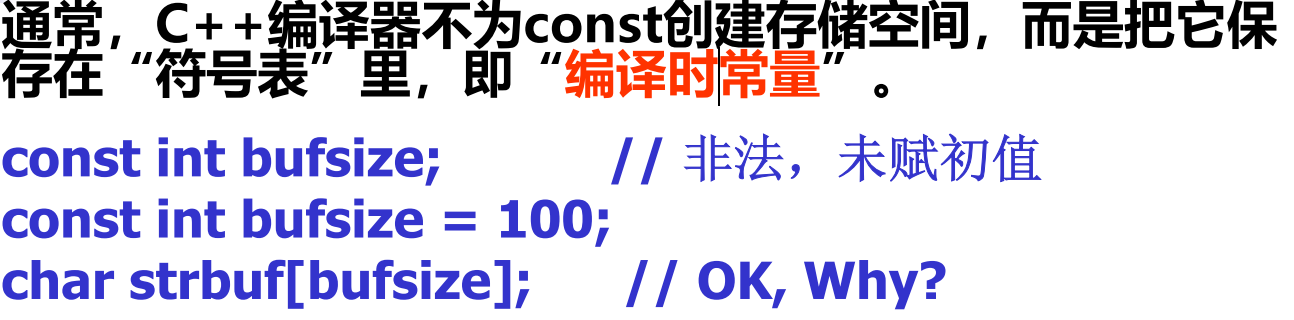


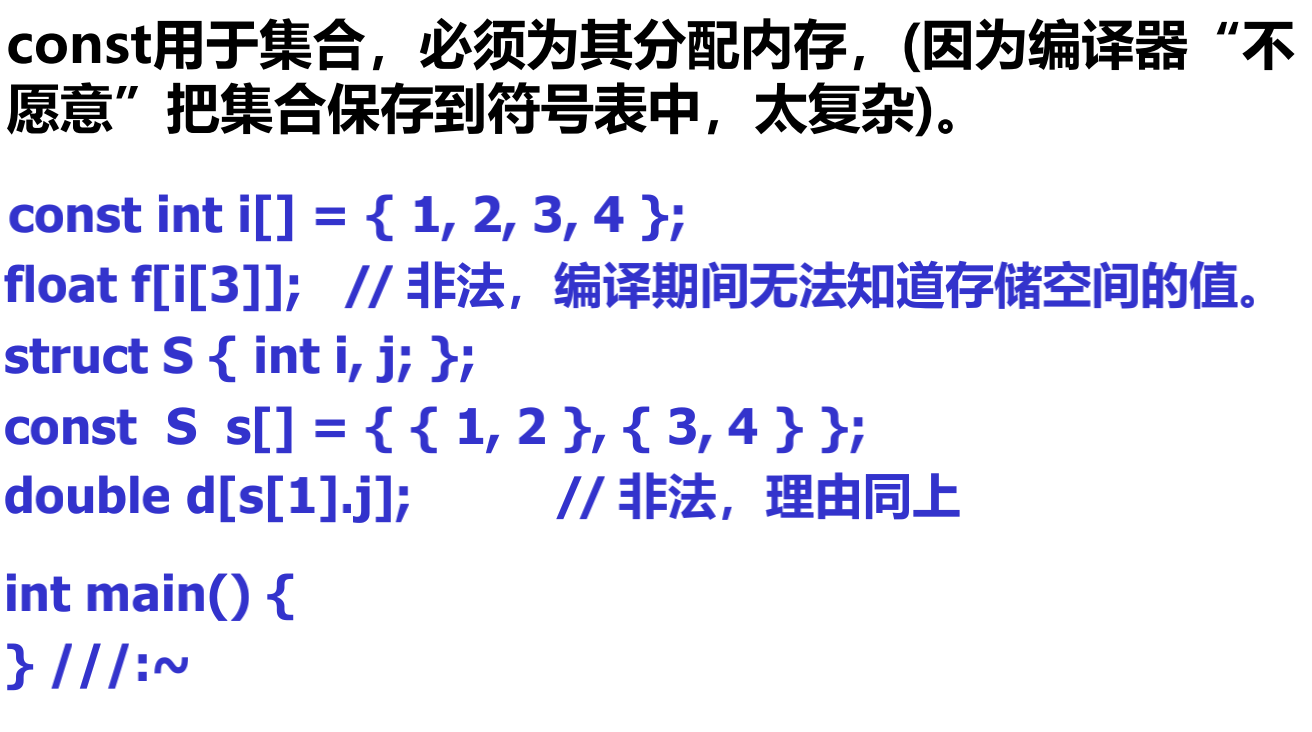


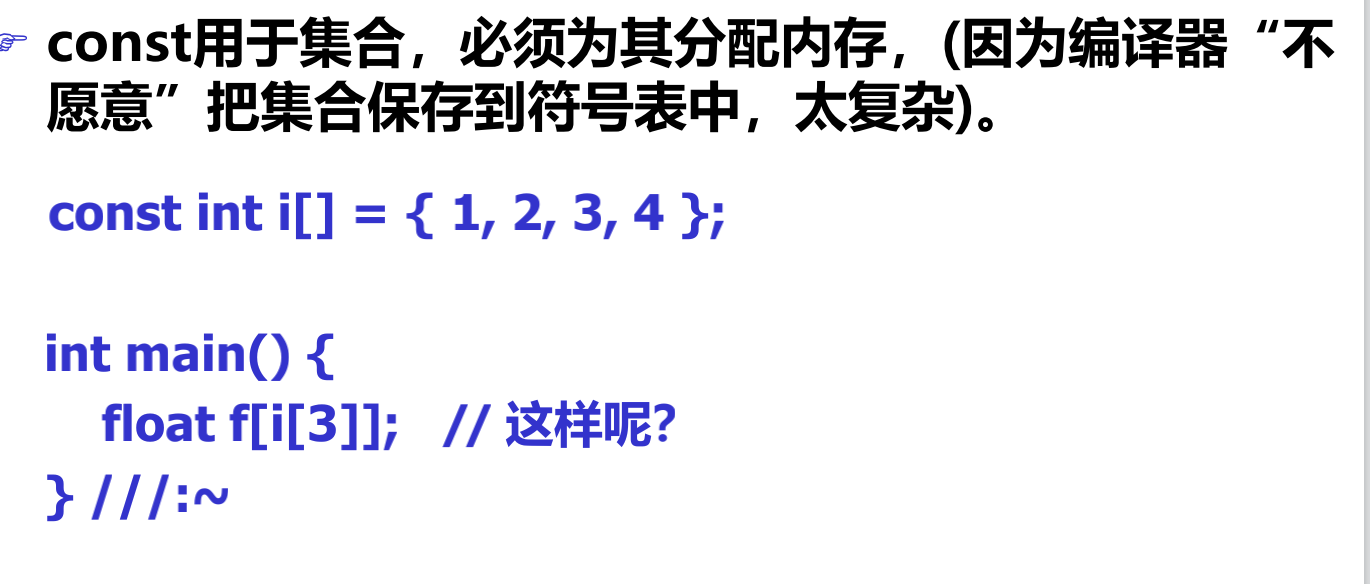


先C C1，再构造

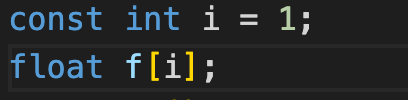
Const 一定要赋初值



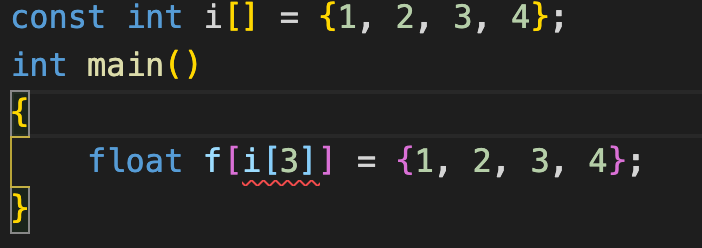




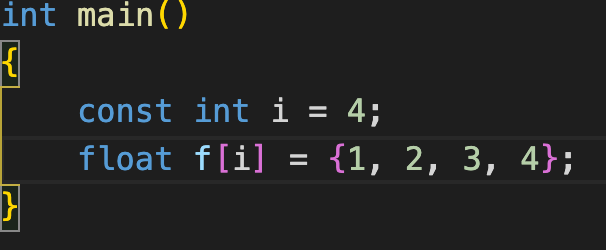
可以，但不能赋初值



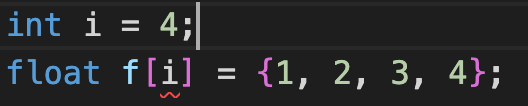
可以



不行

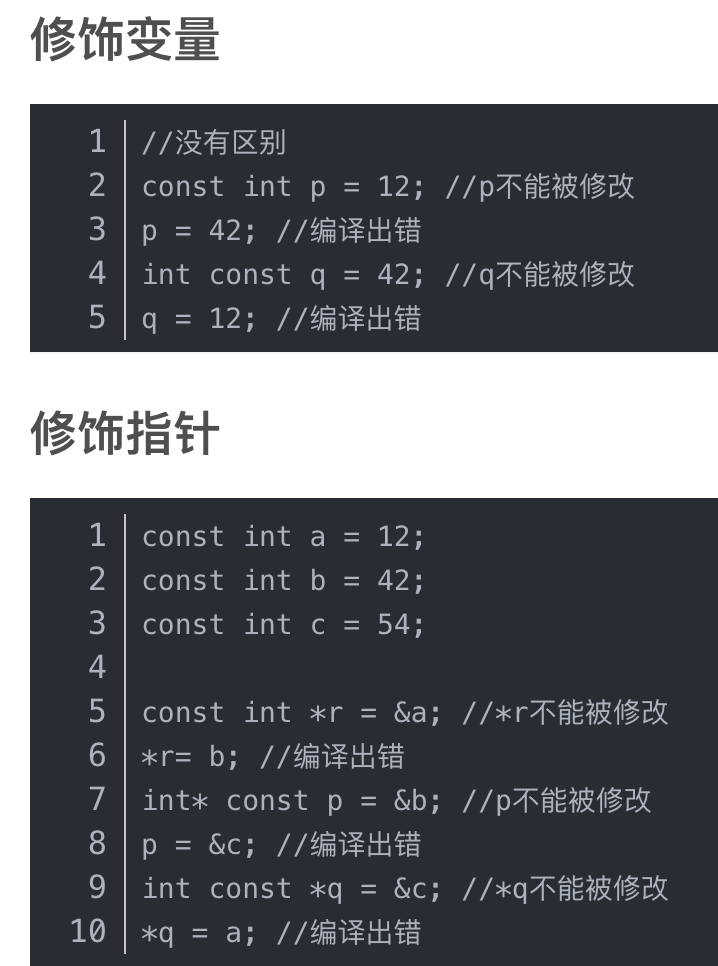


可以



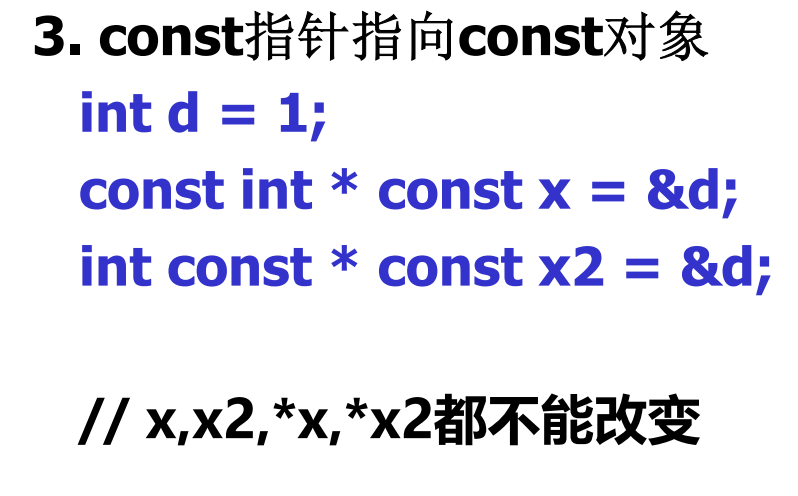
不行

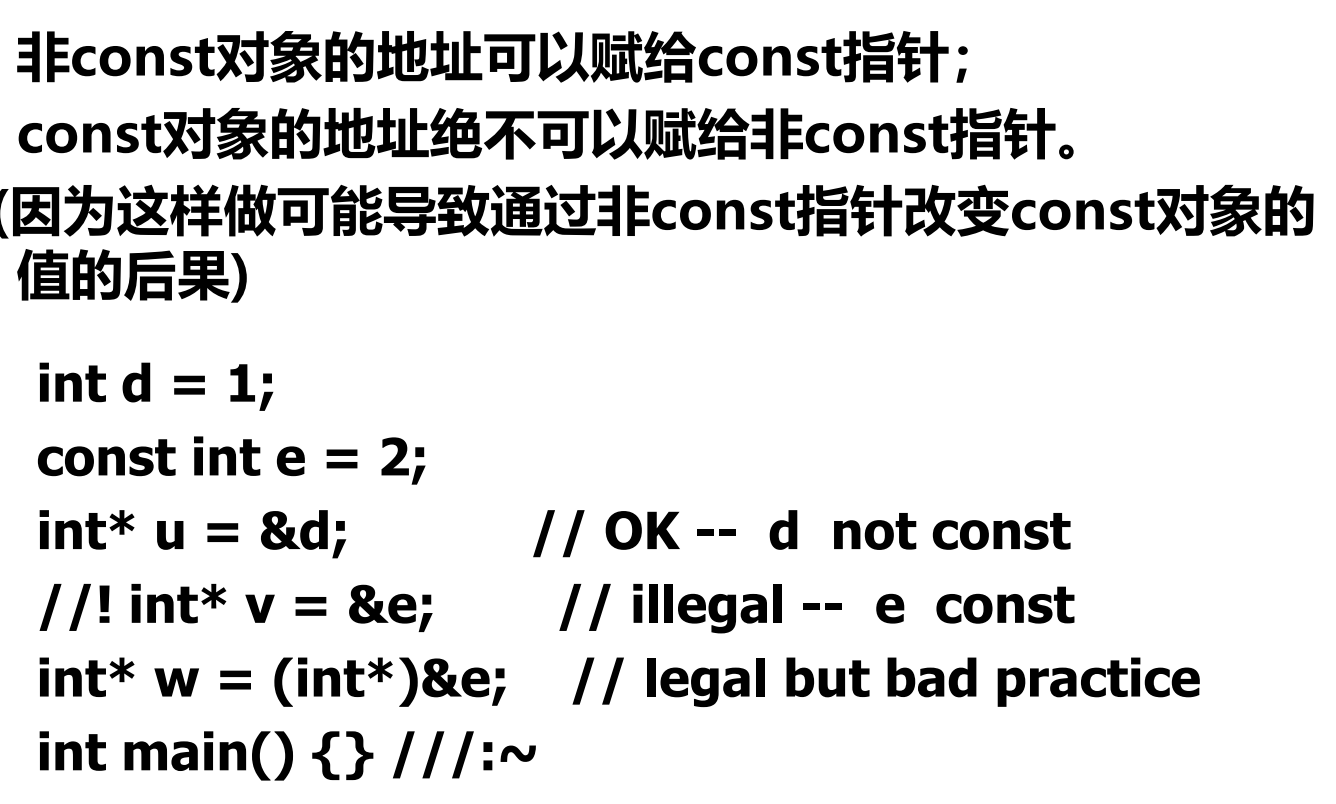
new可以用变量



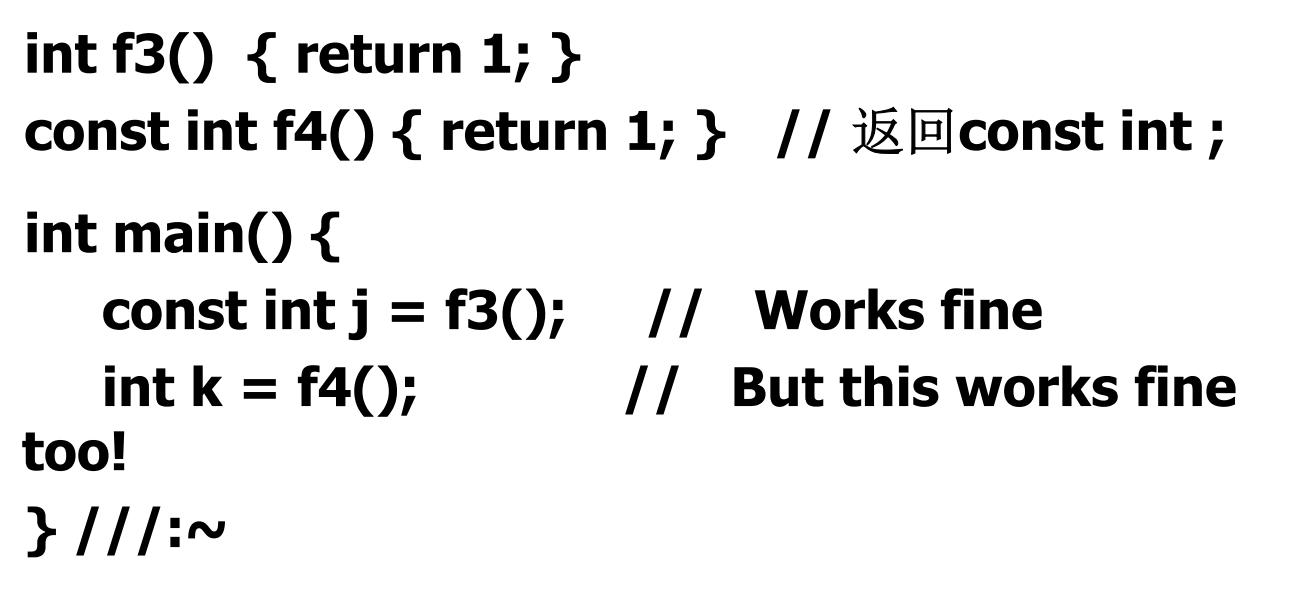
没有const\* int const int和int const等价

对于const我们可以使用“去除类型，就近原则”来理解，即const修饰的是去除类型后最靠近它的右边的那个变量（包含解引用符号\*），

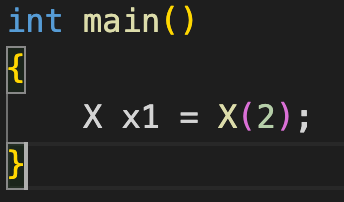




强制转换了，也改不了e的值

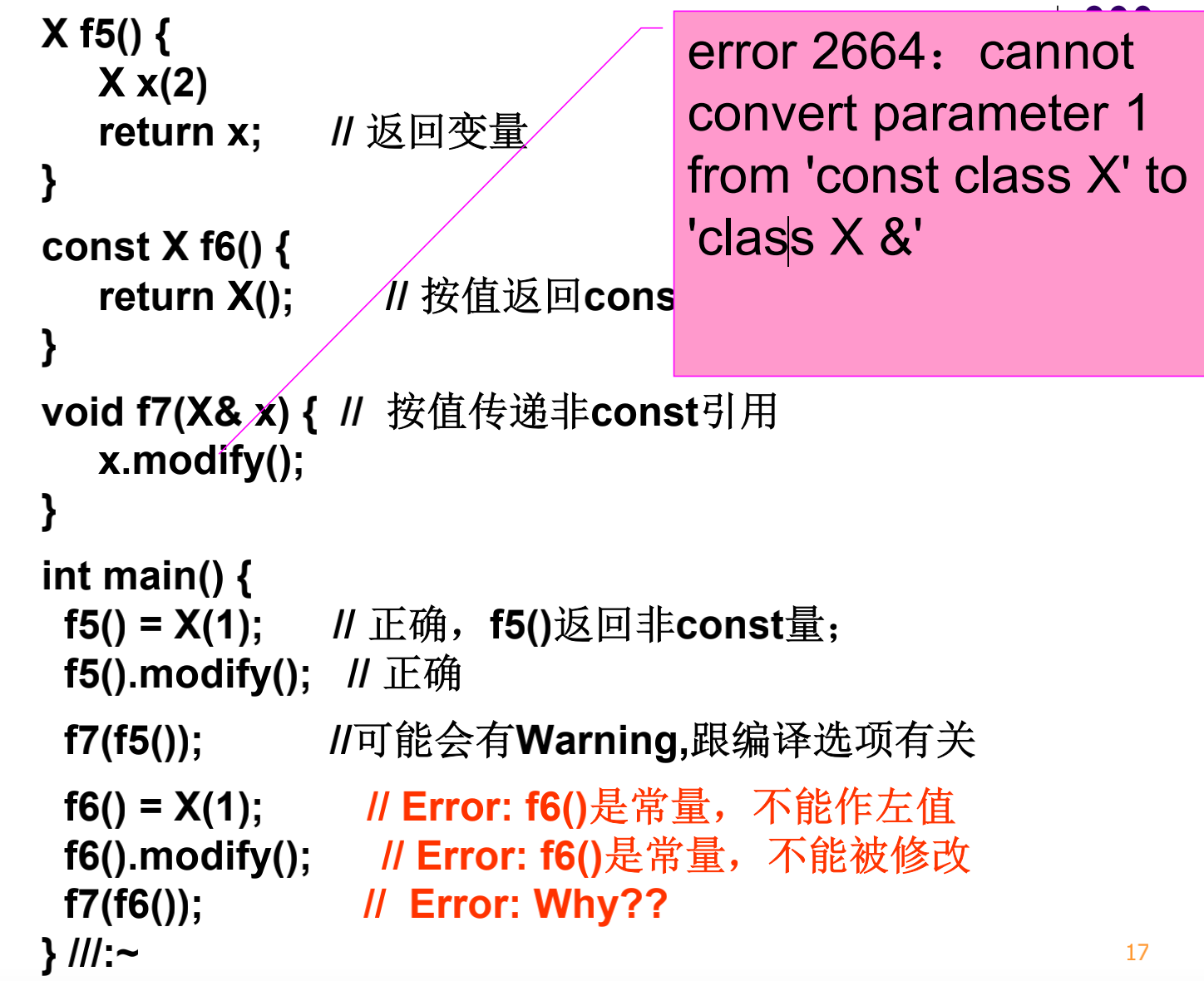


k依然能被修改



和X x1(2)等价

也可以重载了+以后X x1=x2+x3



第一行f5返回的东西可以赋值，但不知道去哪了，再来f5就不一样了

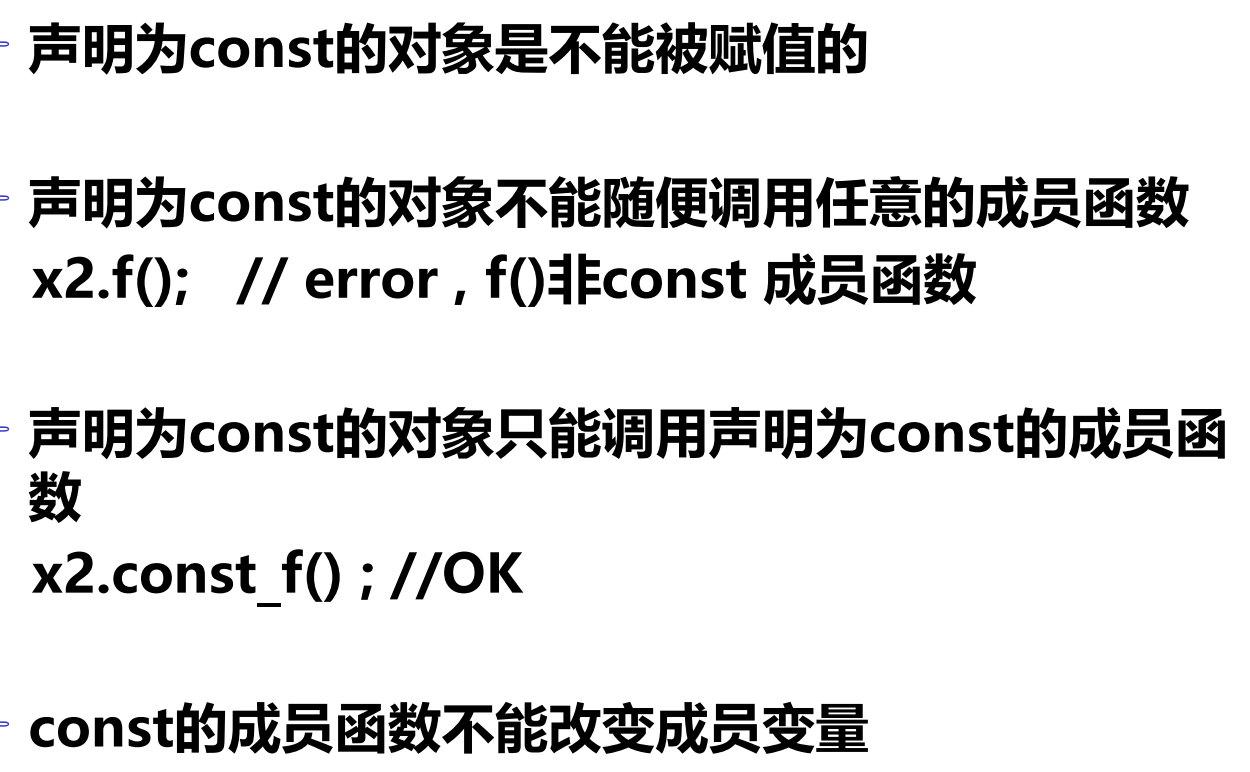
最后一行 no matching function

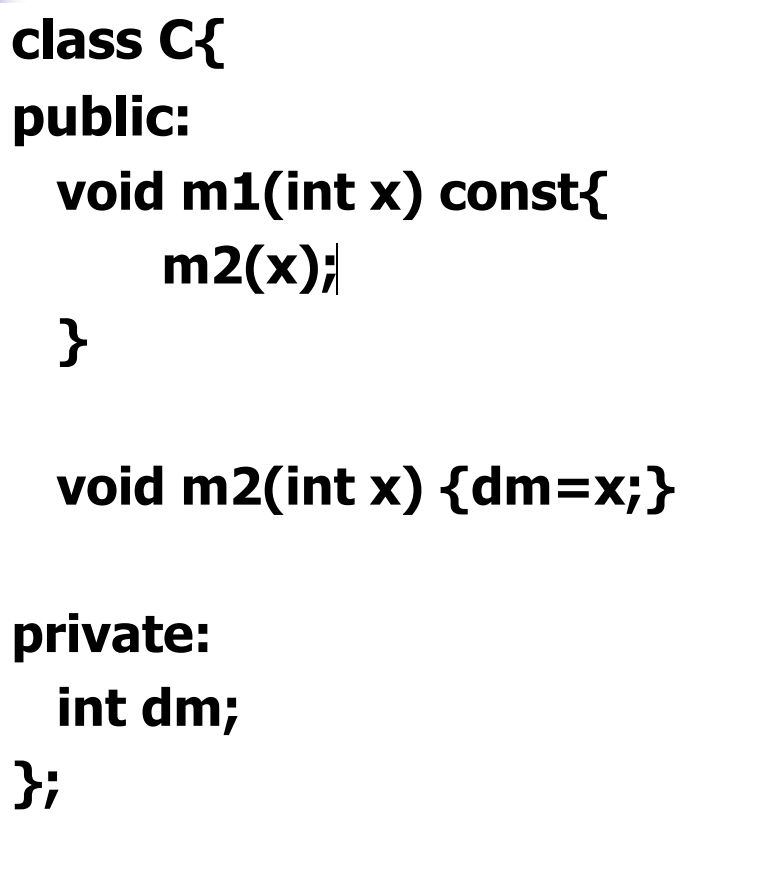
如果是void f7(X x)可以，因为相当于x=const....

Int\*能给const int\*赋值，反过来不行

类数据成员赋初值C++11才行

Const 数据成员在构造函数初始化

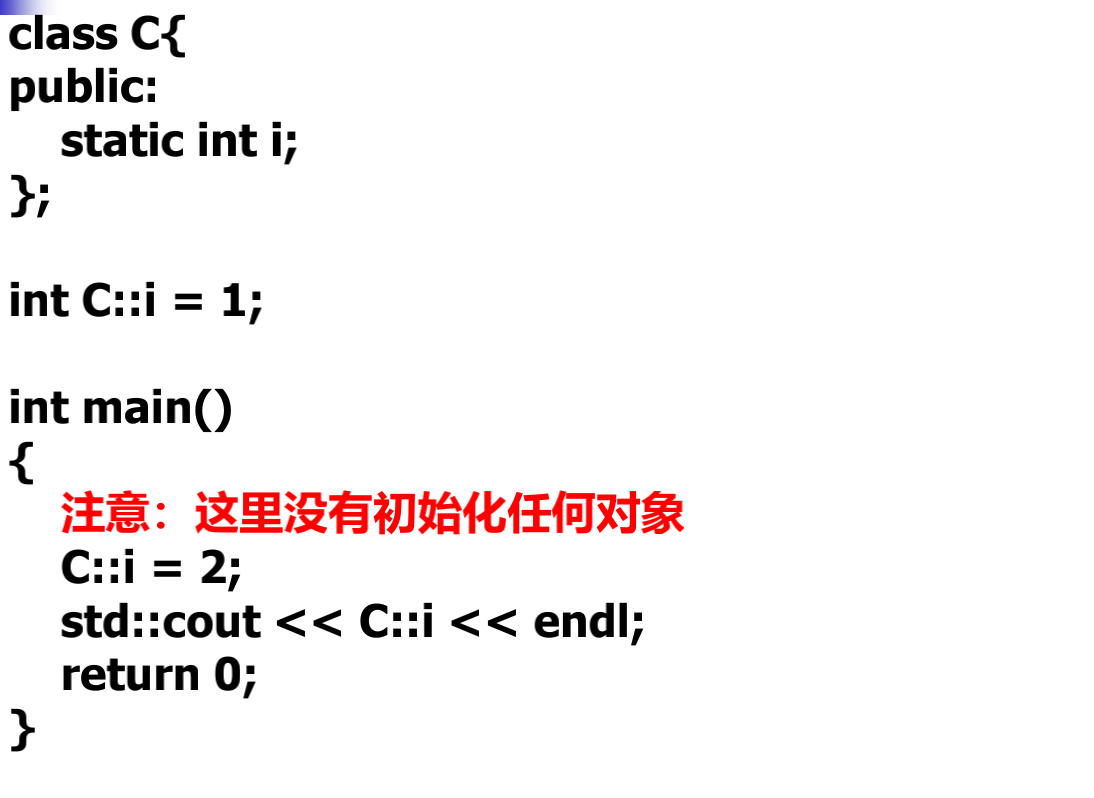




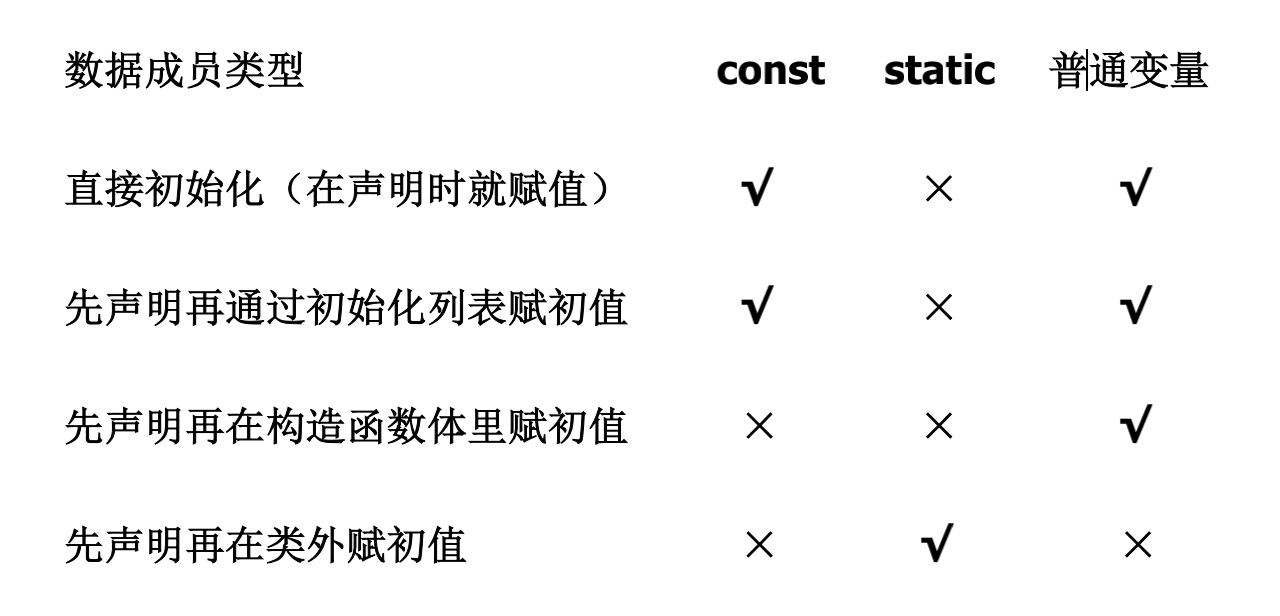
错误



const对象一定要有自己的构造函数。

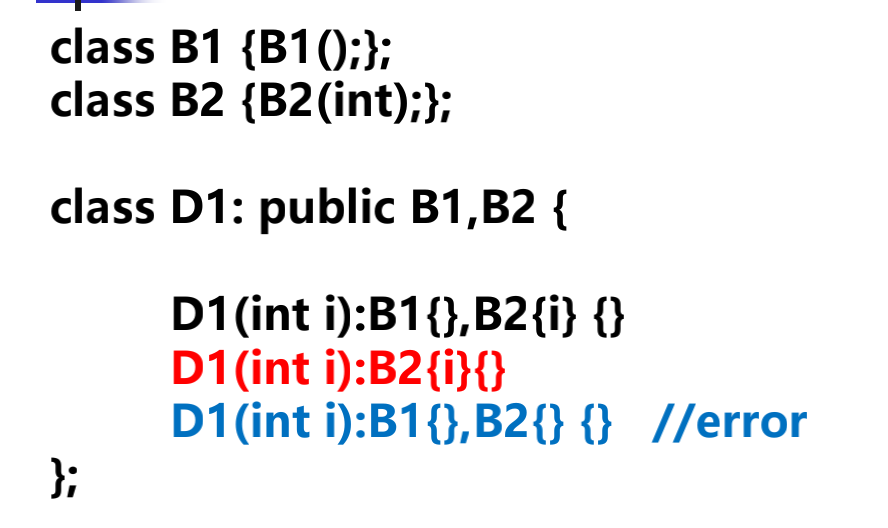


外面要加int，里面不用，没外面不能单有里面



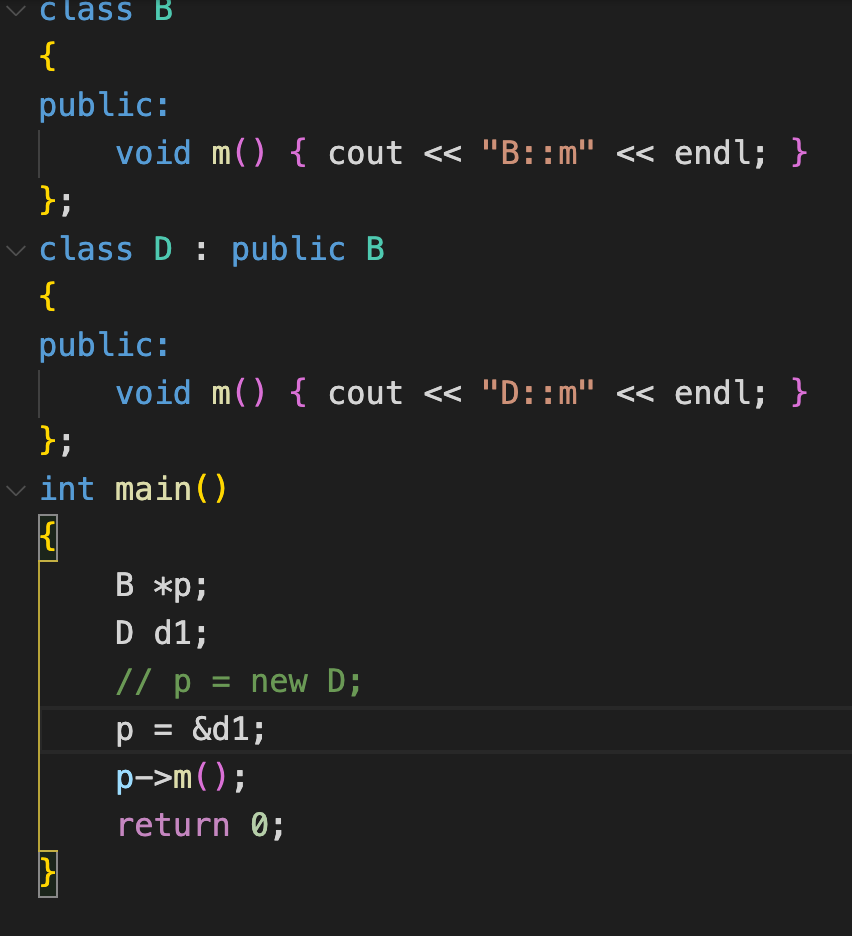




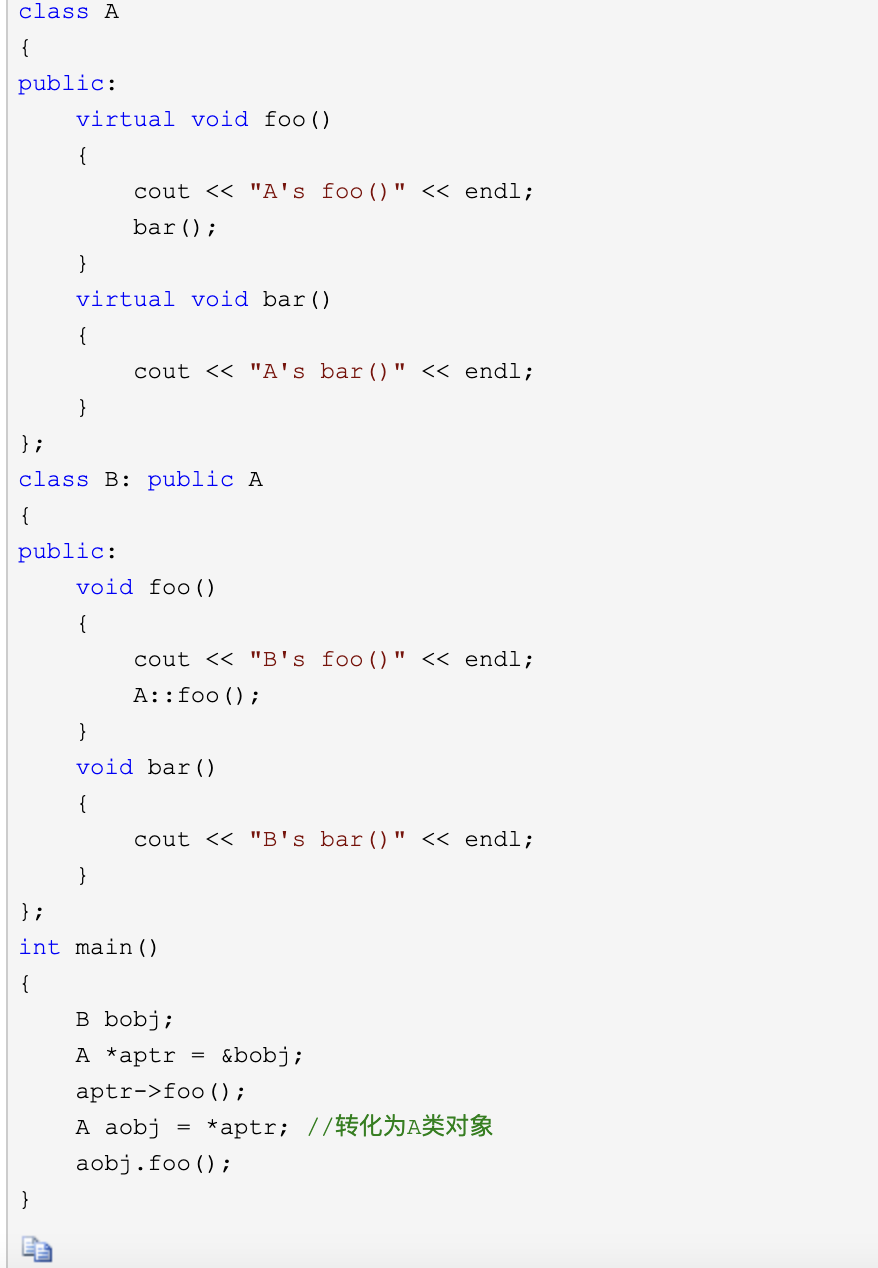


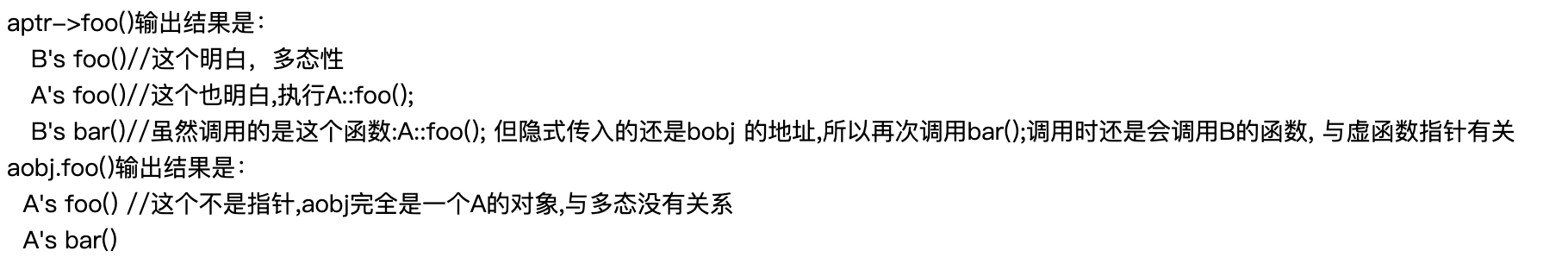
理论上前面B1B2要加public，{i}应为()

1，如果以一个基础类指针指向一个衍生类对象(派生类对象)，那么经由该指针只能访问基础类定义的函数（静态联翩）  
2，如果以一个衍生类指针指向一个基础类对象，必须先做强制转型动作（explicit cast），这种做法很危险，也不符合生活习惯，在程序设计上也会给程序员带来困扰。（一般不会这么去定义）  
3，如果基础类和衍生类定义了相同名称的成员函数，那么通过对象指针调用成员函数时，到底调用那个函数要根据指针的原型来确定，而不是根据指针实际指向的对象类型确定。这里是因为原来是虚函数！！！全部普通继承的话由指针类型决定

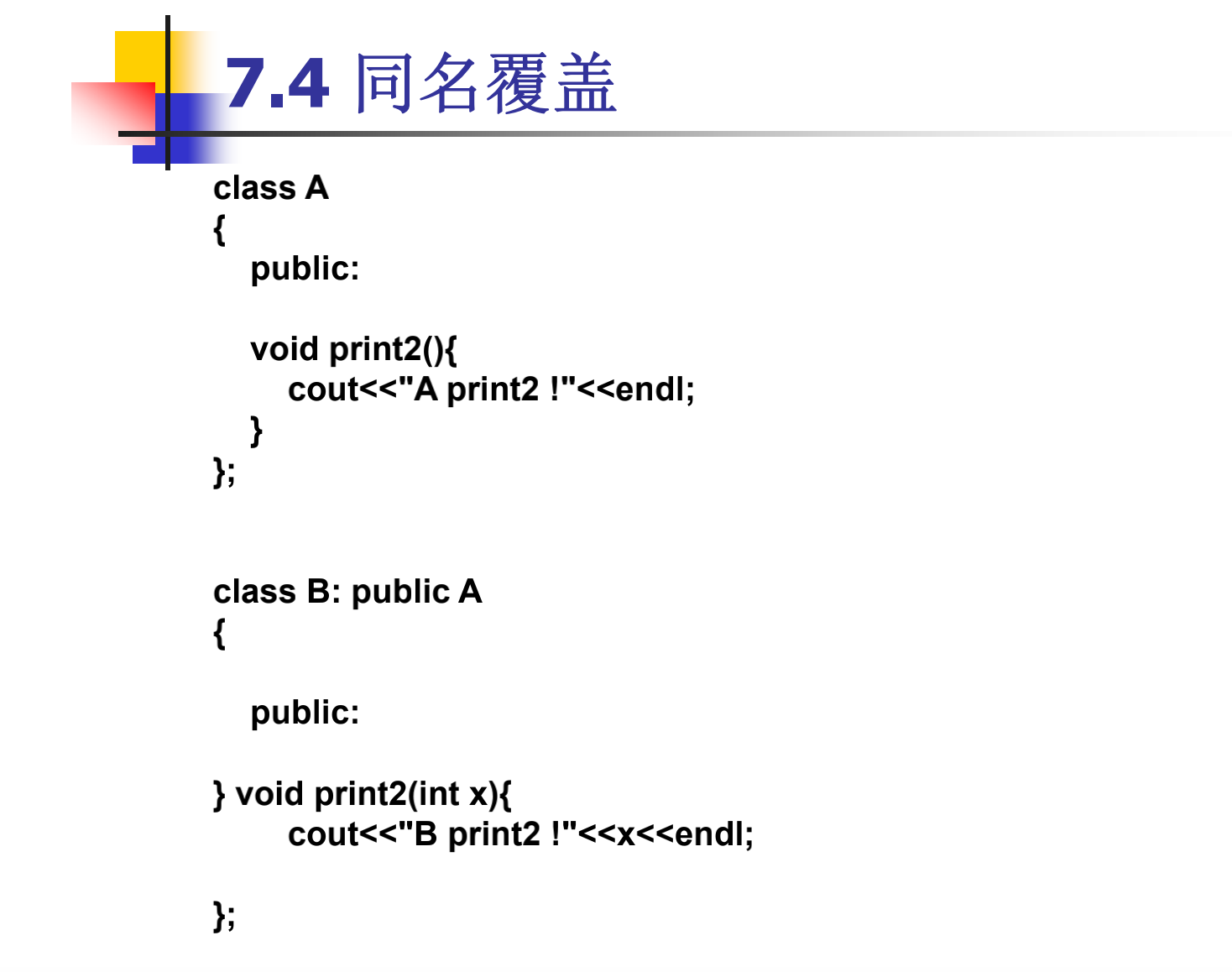


B::m



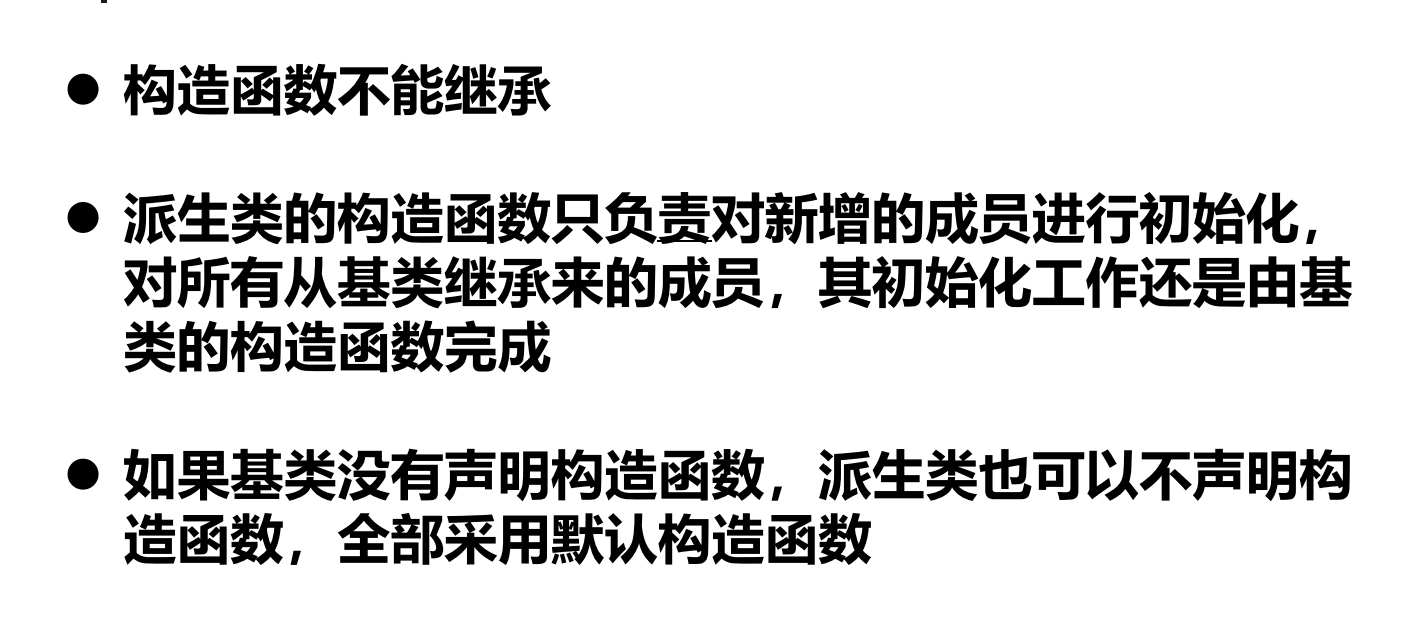


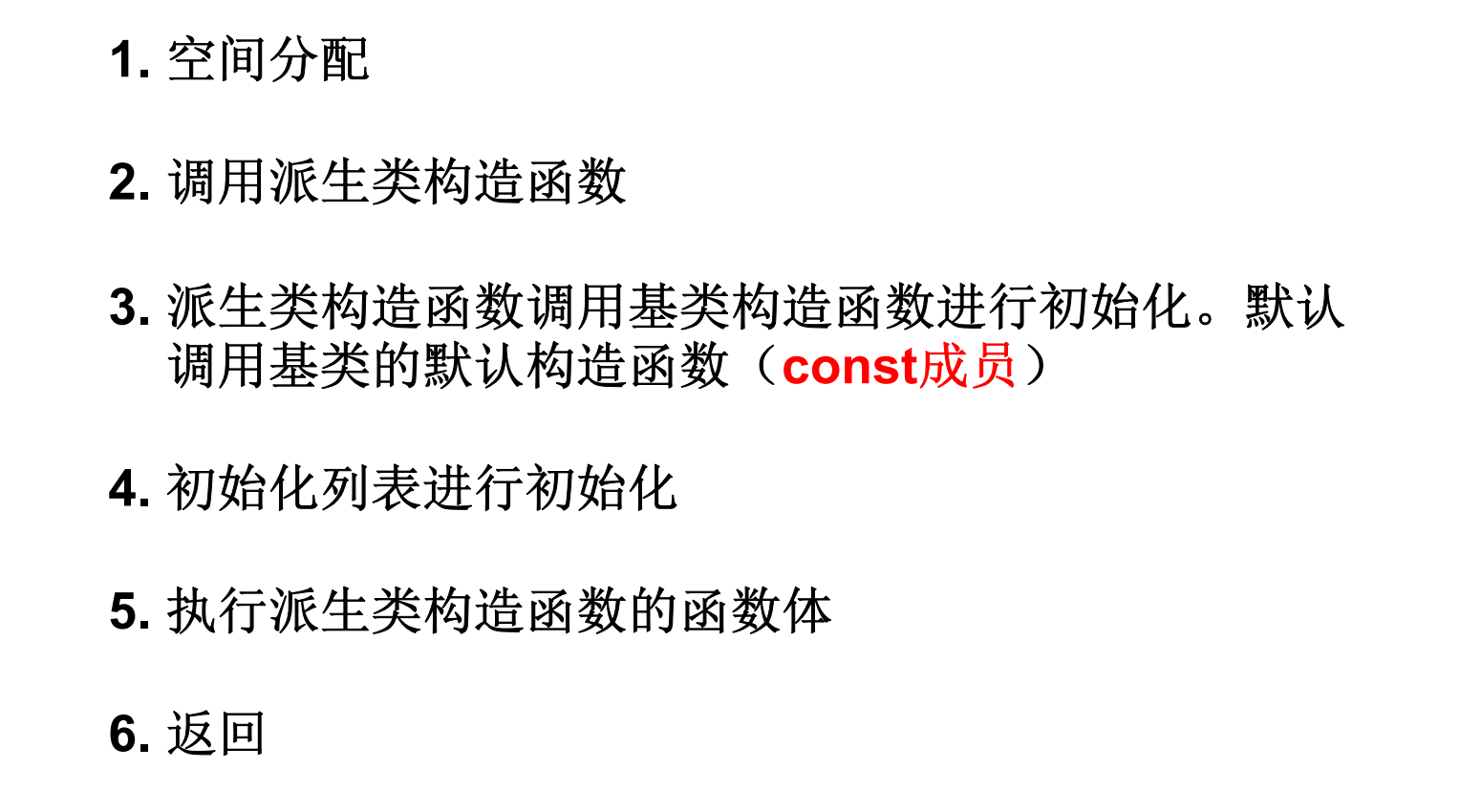
第一个是因为他本来是b，所以调用b的



A的print2没了，否则要加using A::print2 《-没括号

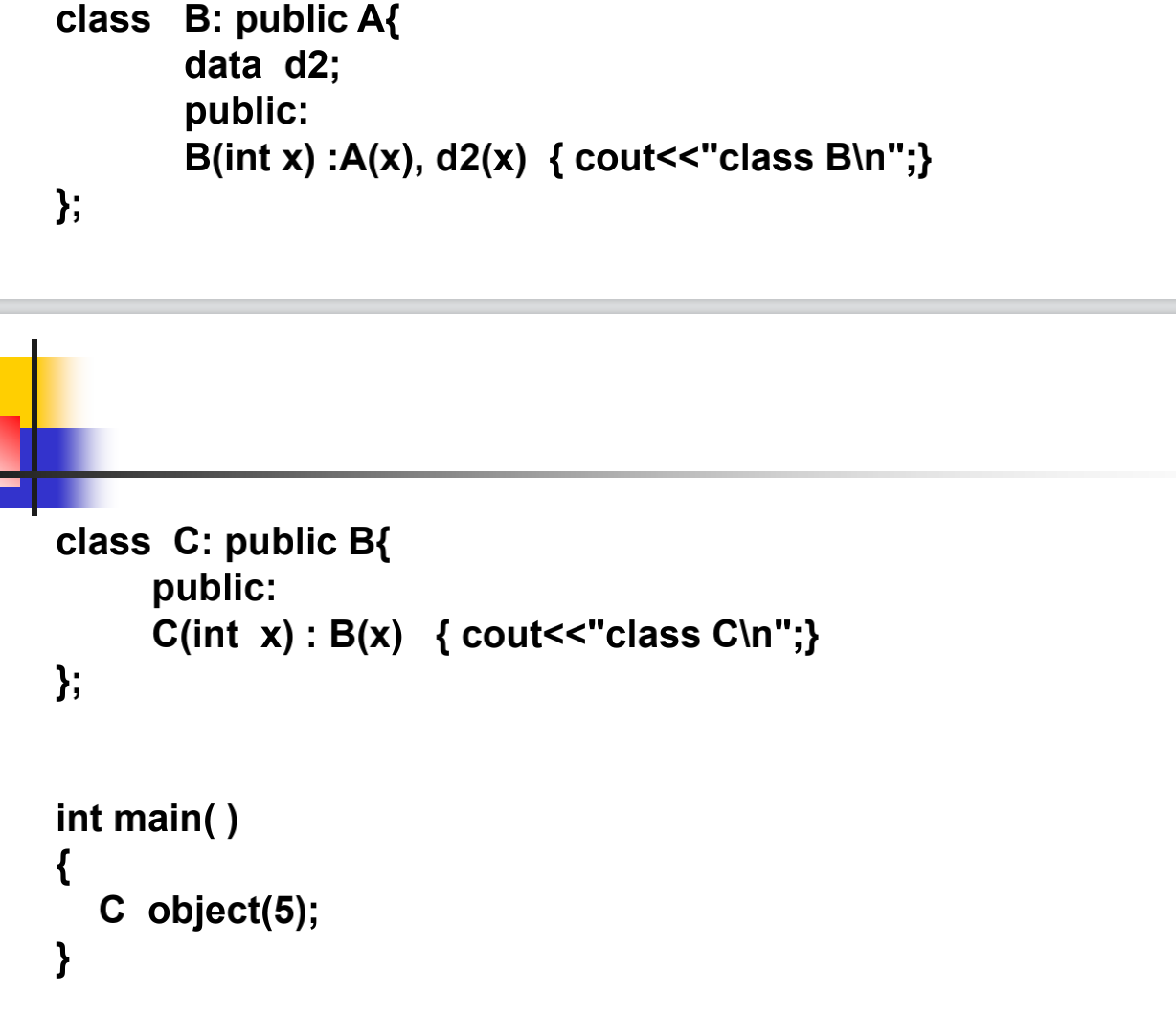
B::A.print2()





先基类

基类不含参数的构造函数不用写出来



这里A和d2换个顺序也一样，因为都是父类-》成员对象-》自己

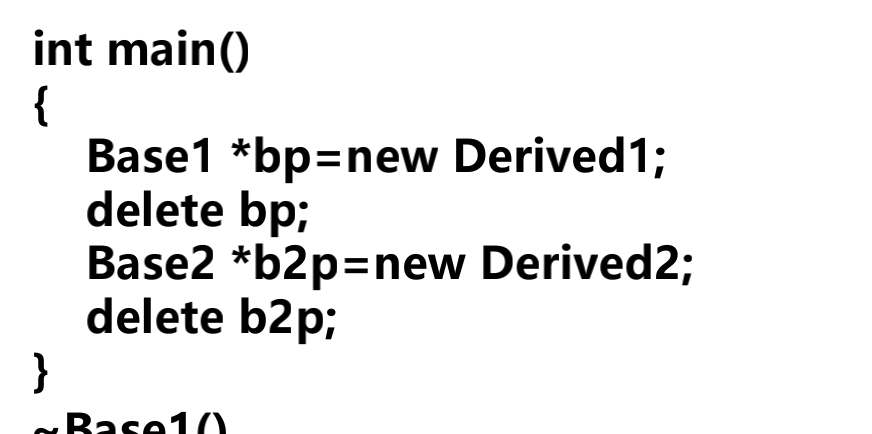


依然先A后data

清理：自己-》对象-》父类



不是，不能



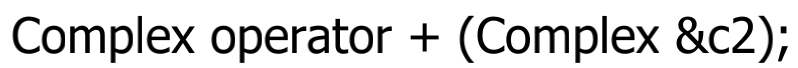
父类型指针指向子，不是虚函数用父的，是虚函数用子的。函数传递用哪个函数看的是指针类型（前面那个）

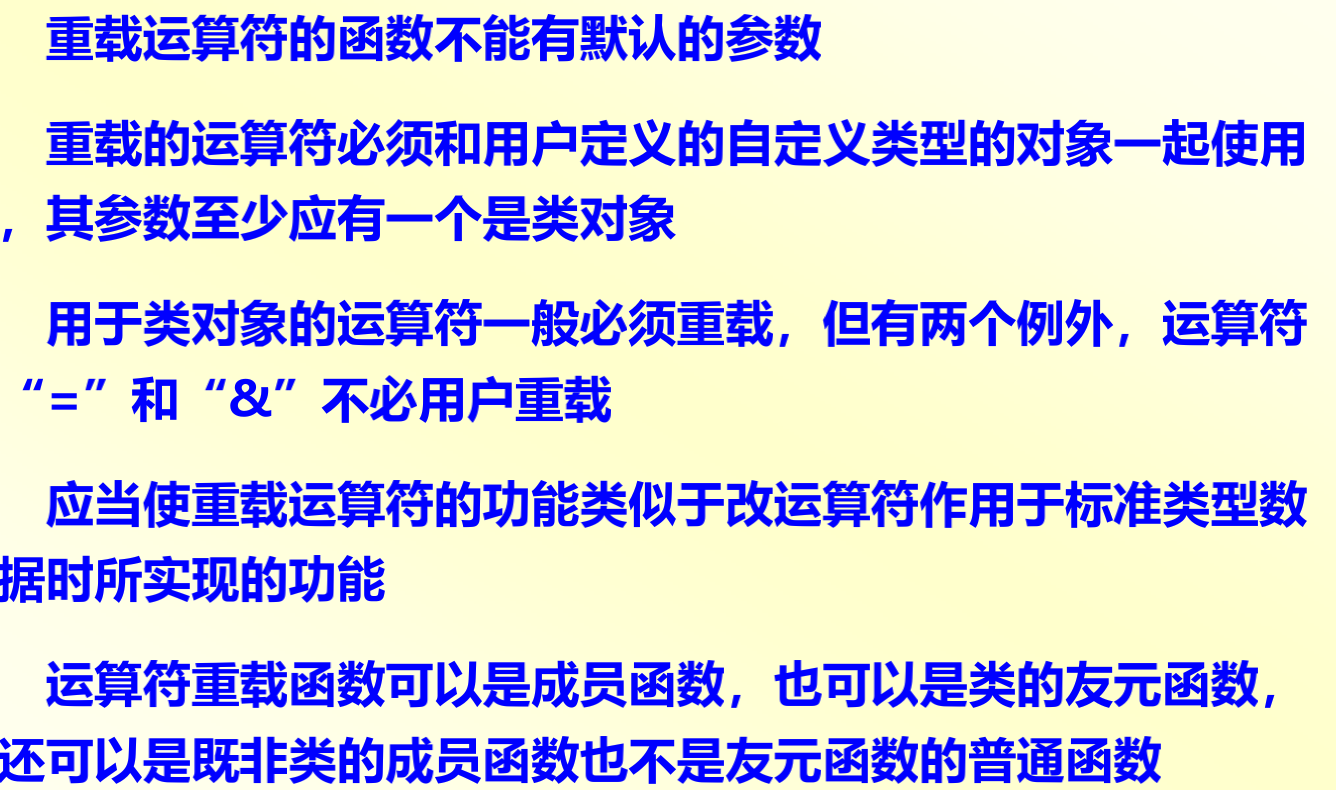
有虚函数就是看指向的东西类型，没有看他本身的类型

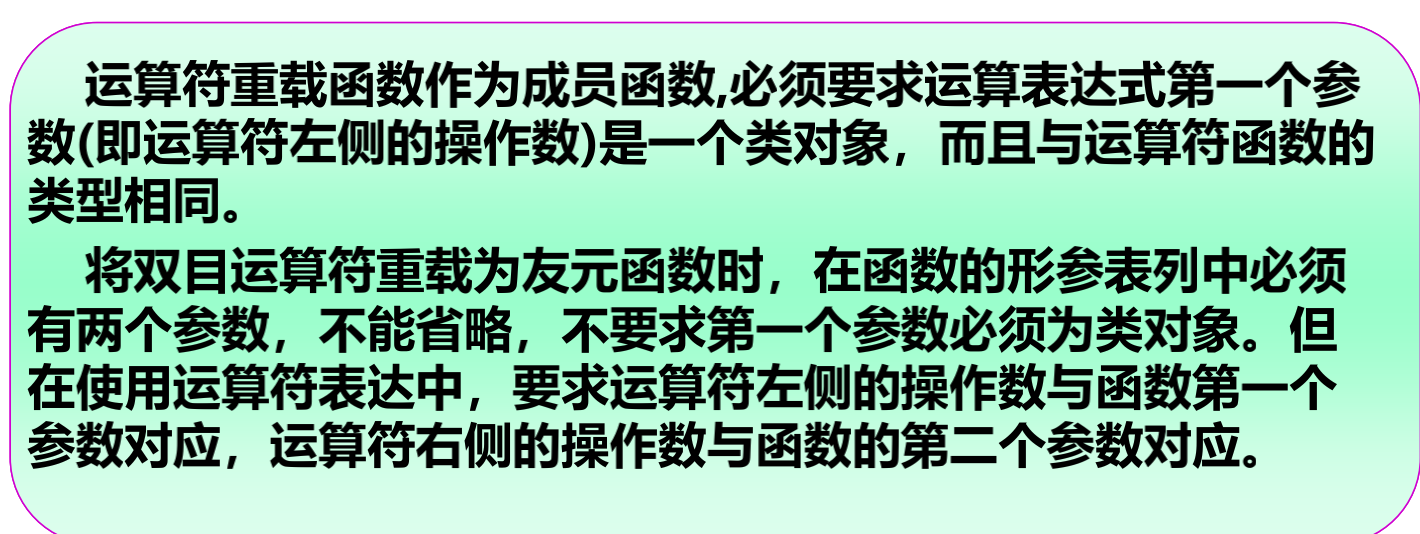
带virtual纯虚函数是抽象类（virtual.....=0），自己不能有实例。不是纯虚的可以有实例。

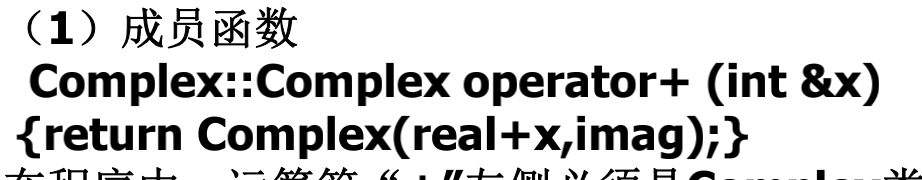
纯虚函数必须被重写，否则继承出来的子类也是抽象类。

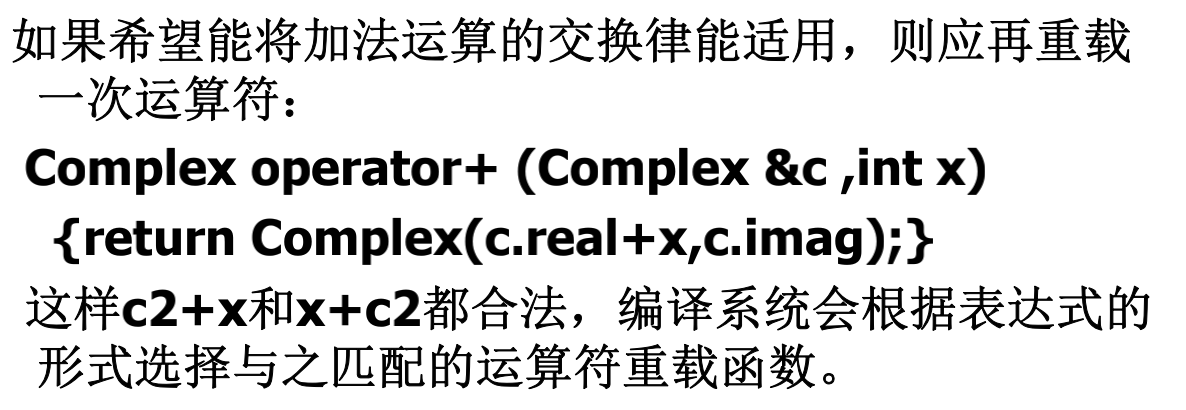
P->m()有括号



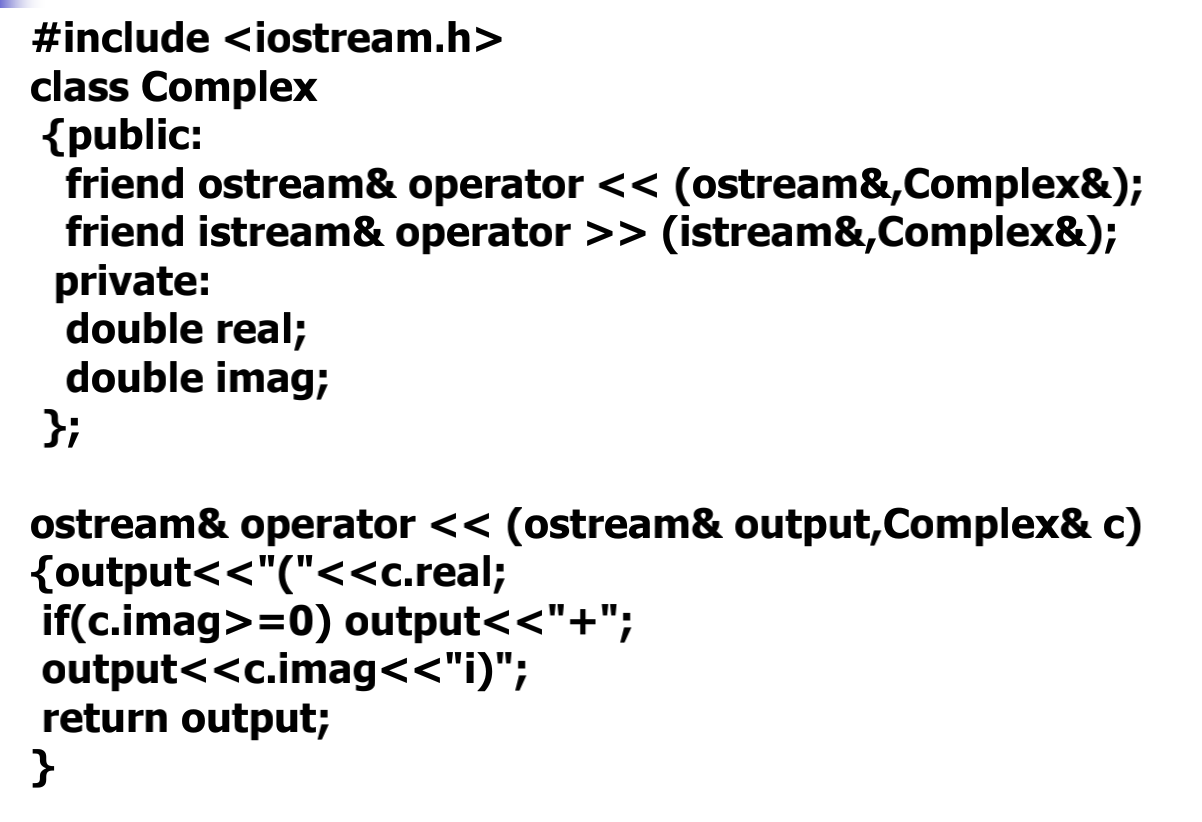














把int全改X X x1=X(0.5);

转换构造函数：别的变类

类型转换函数：类变别的



函数模板不允许自动类型转换  
普通函数能够进行自动类型转换

同名变量、同名函数同时存在，先调用自己的

类赋值不会调用构造函数

继承类必须构造不是默认构造的父类，虚的也得写（即使没用）

private的static也可以在外面赋值

重载=返回引用减小内存，+不行

返回引用给一个不是引用的传值，当然没问题

拷贝：构造新的p1=p 形参

=运算：原有的p1=p 返回