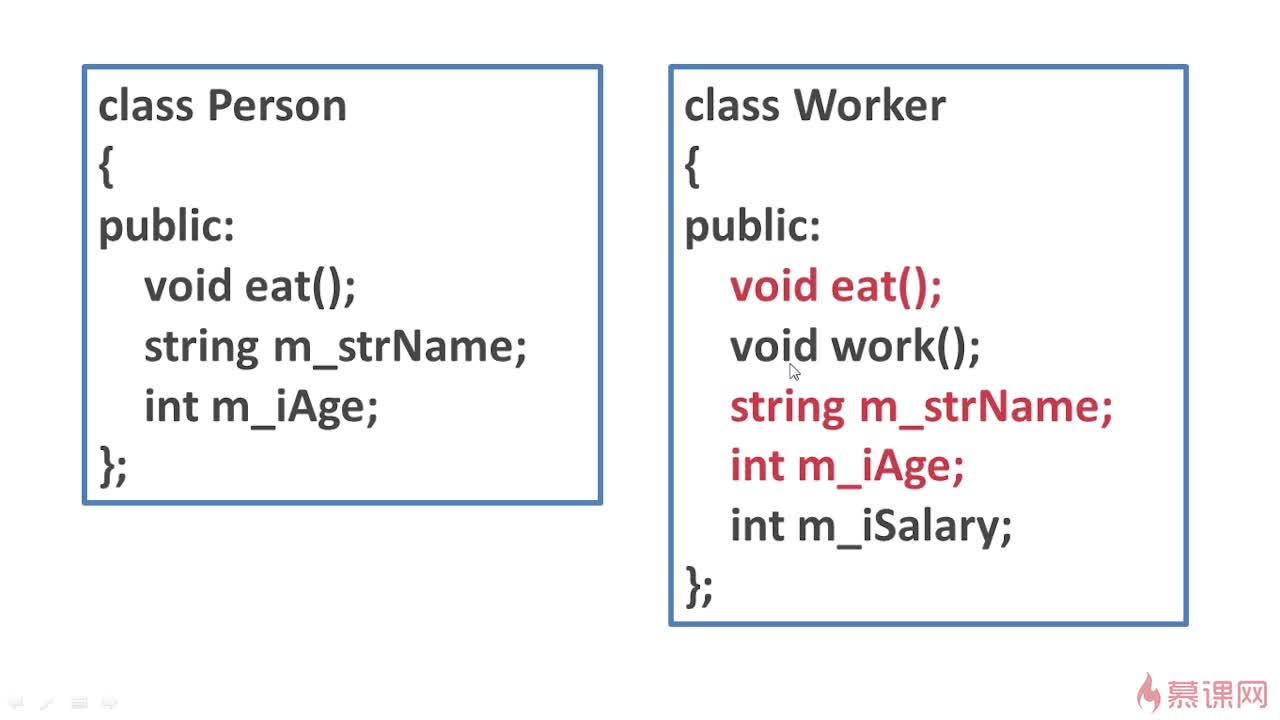
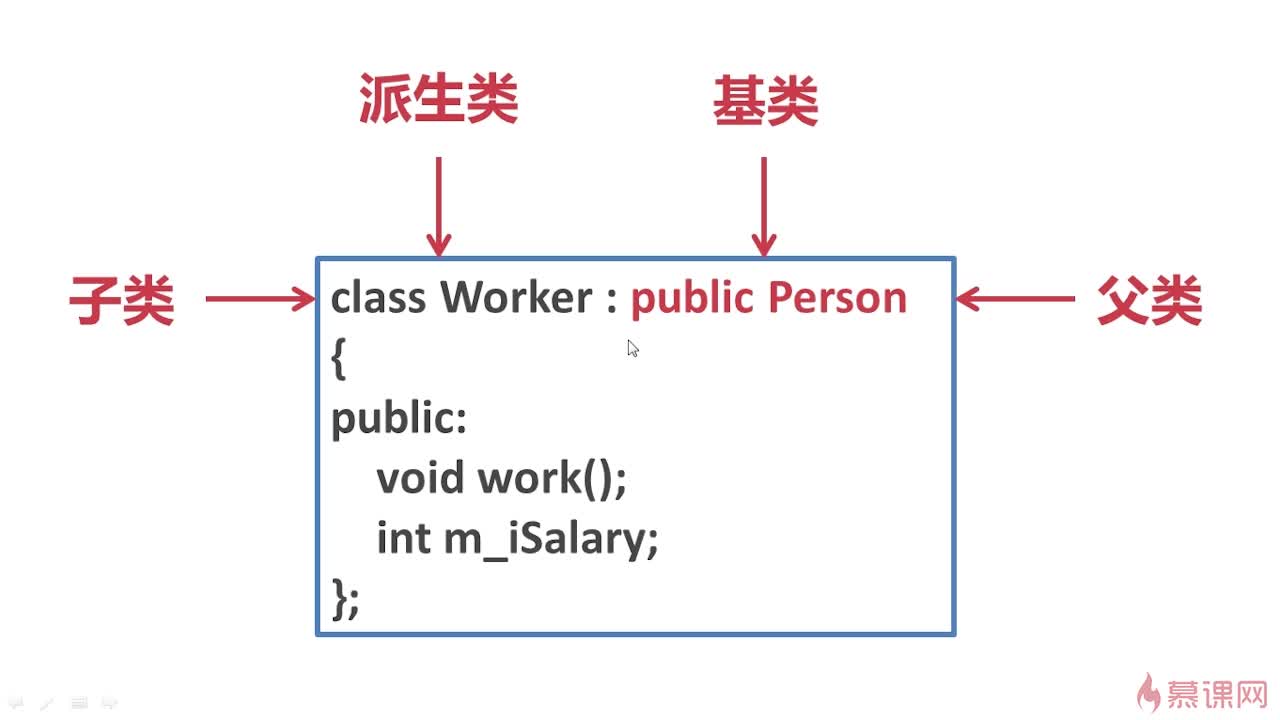
# 1：继承的概述

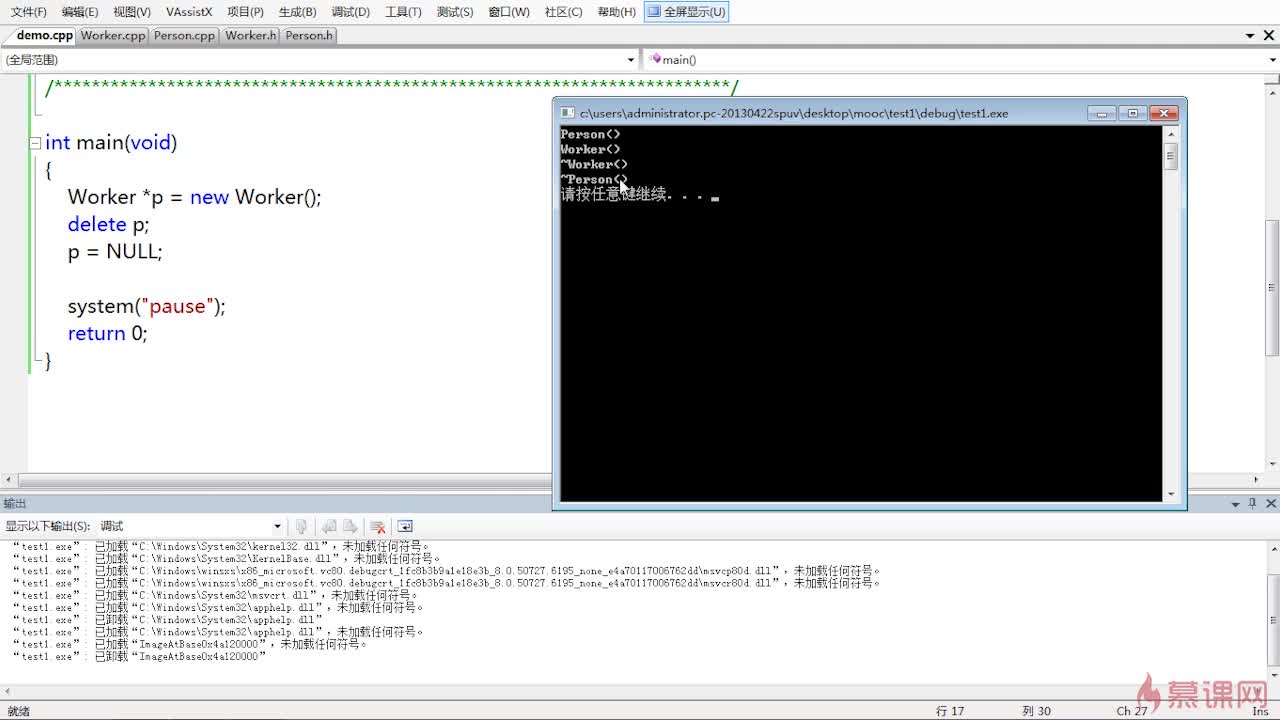
****

****

**基类就是父类，派生类就是子类**

**实例化子类的时候先实例化父类，销毁的时候先销毁子类的析构函数再执行父类的析构函数，这跟对象成员的创建和销毁相似，先创建对象成员再组装新的对象，但是销毁的时候，先销毁新的对象，再销毁对象成员。**

**子类可以访问父类个自身的数据成员和成员函数：**

****

**注意：**

**被继承的类叫做基类也叫做父类，从其他类继承而来的类叫做派生类也叫做子类。**

**子类中不仅继承了父类的中的数据成员，也继承了父类的成员函数。**

**C++中的继承关系是概念上的父子关系，不是个体的父子关系。**

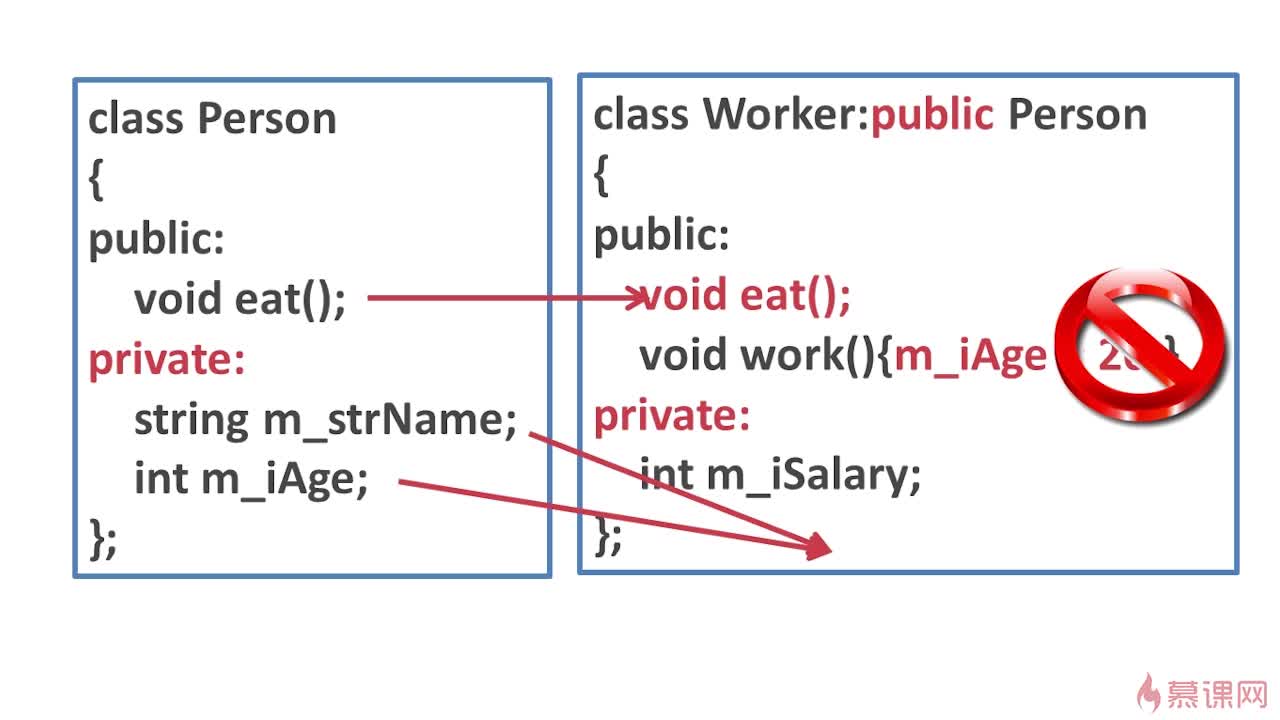
**类与类之间必须遵循概念上的父子关系，否则将造成定义和使用的混乱**

## 三种继承方式：

****

### public继承：

**public 公有继承下 private访问限定符**

****

**Public共有继承下private包含的内容将会被继承到子类的未知区域（注意不是private）部分（注意，不是没有继承，而是继承到了无法直接访问的区域），所以public继承方式下子类无法访问父类的private部分。**

**对比：public公有继承下 protected访问限定符 可操作成员及函数**

### protected继承和private继承：

****

****

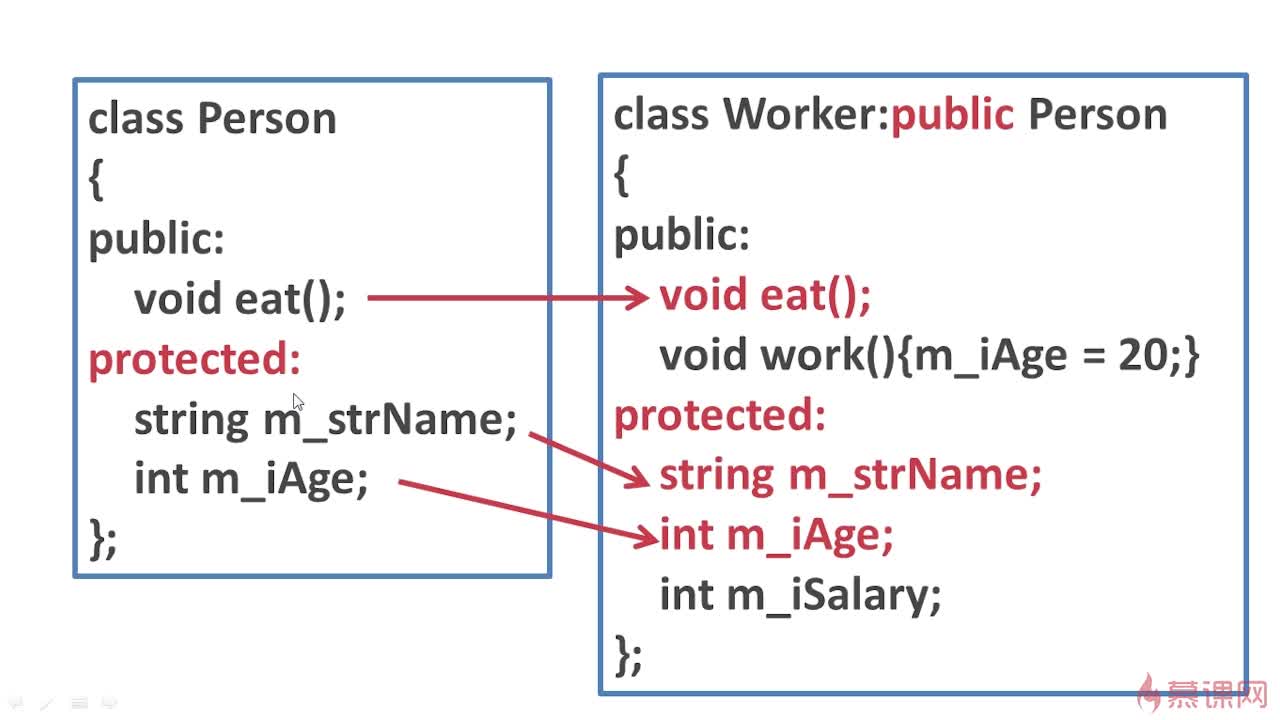
**对比三种继承方式我们可以得出以下结论：**

**派生类private三种继承都无法访问**

**public继承后各归各**

**protect继承后全为protect**

**private继承后也全为private**

****

**所以可以得出以下结论：private在不发生集成的情况下和protected完全等价，但是二者的区别就在于继承时的效果。**

**public继承方式下各种性质（对堆和栈实例化对象都具有适用性）**

****

**总结：person当中的protected的数据成员被继承到worker的protected限定符中**

**person当中的private的数据成员无法被继承到worker的private限定符中**

**父类的保护成员继承到子类的保护成员下面，可以通过子类的成员函数访问，不能外部直接访问；**

**父类的私有成员无法继承到子类，也就无法通过子类访问**

**总结：**

**B类从A类派生，那么B类是A类的子类，A类是B类的超类。**

**B类中含有A类的所有数据成员。 只是 private 成员不可访问**

**B类从A类公共派生，那么可以通过B类的对象调用到A类的公共及保护限定符下的成员函数**

**可以在B类中直接使用A的公共及保护限定符的数据成员。**

**A类的公共成员函数成为B类的公共成员函数。**

**A类的保护成员函数成为B类的保护成员函数。**

**A类的私有成员函数不能被B类直接使用（但是也继承了）。**

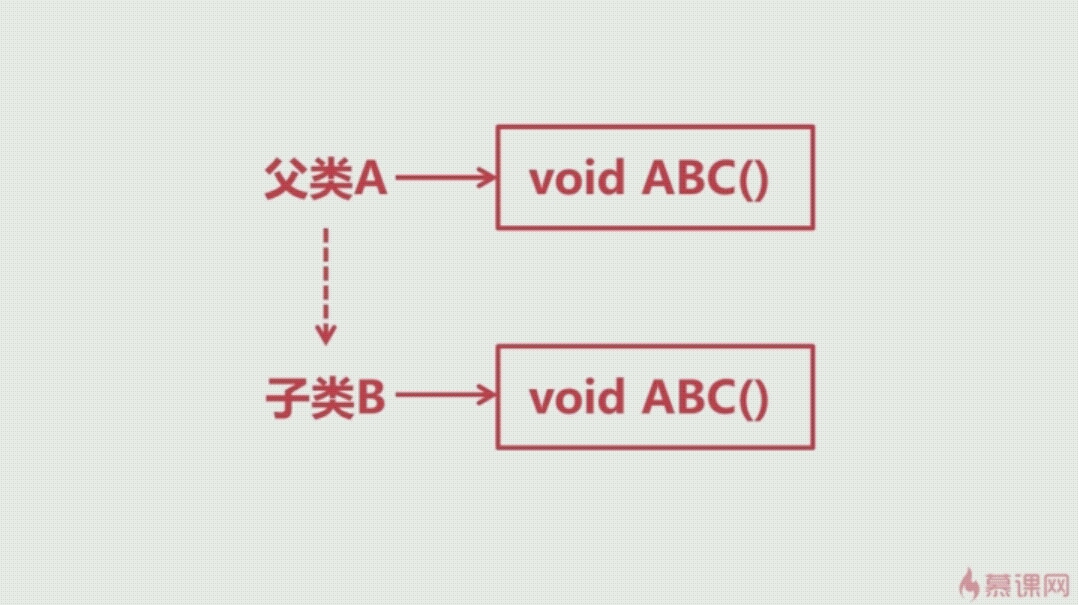
**B类从A类私有派生，那么A类的公共成员函数成为B类的私有成员函数。**

**B类从A类保护派生，那么A类的公共成员函数成为B类的保护成员函数。**

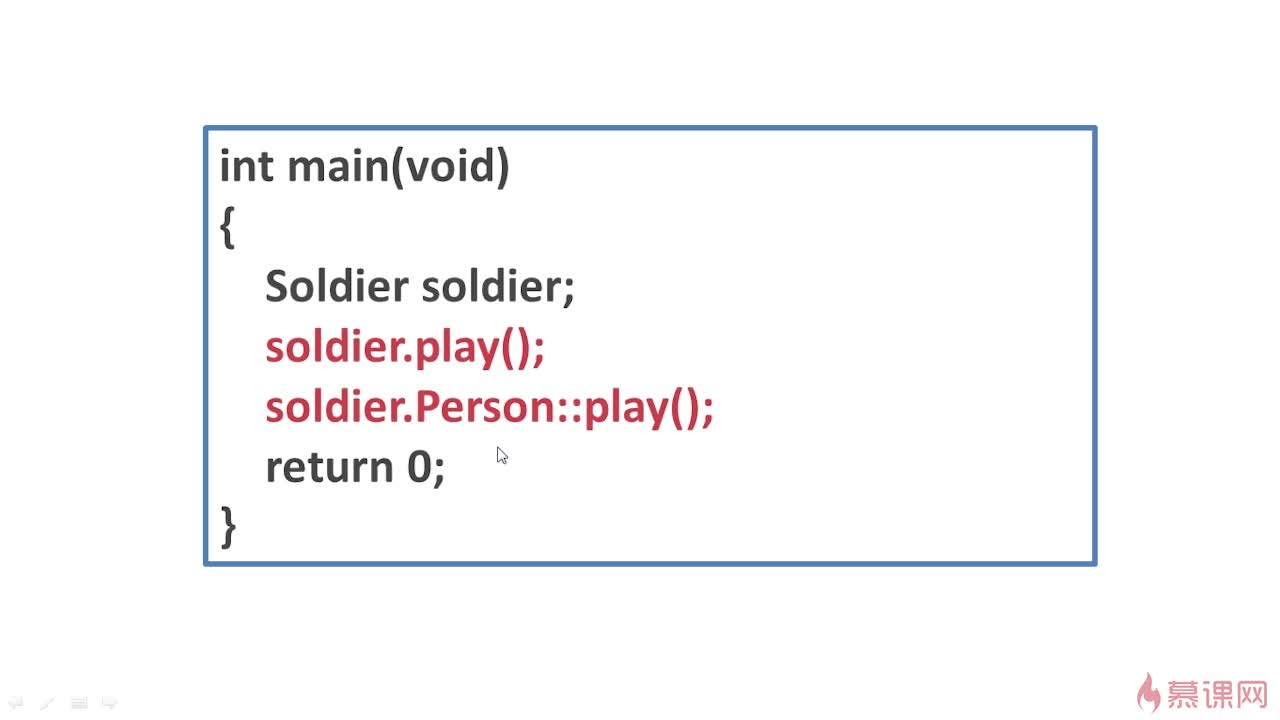
**A类的保护成员函数成为B类的保护成员函数。**

## 隐藏：

**从父类中继承void ABC()，子类中也存在void ABC()时，父类继承的函数将被隐藏**

****

**父子关系有同名函数或变量时，父类将自动隐藏，若想调用父类中的成员函数或者数据成员可以用以下方式：**

****

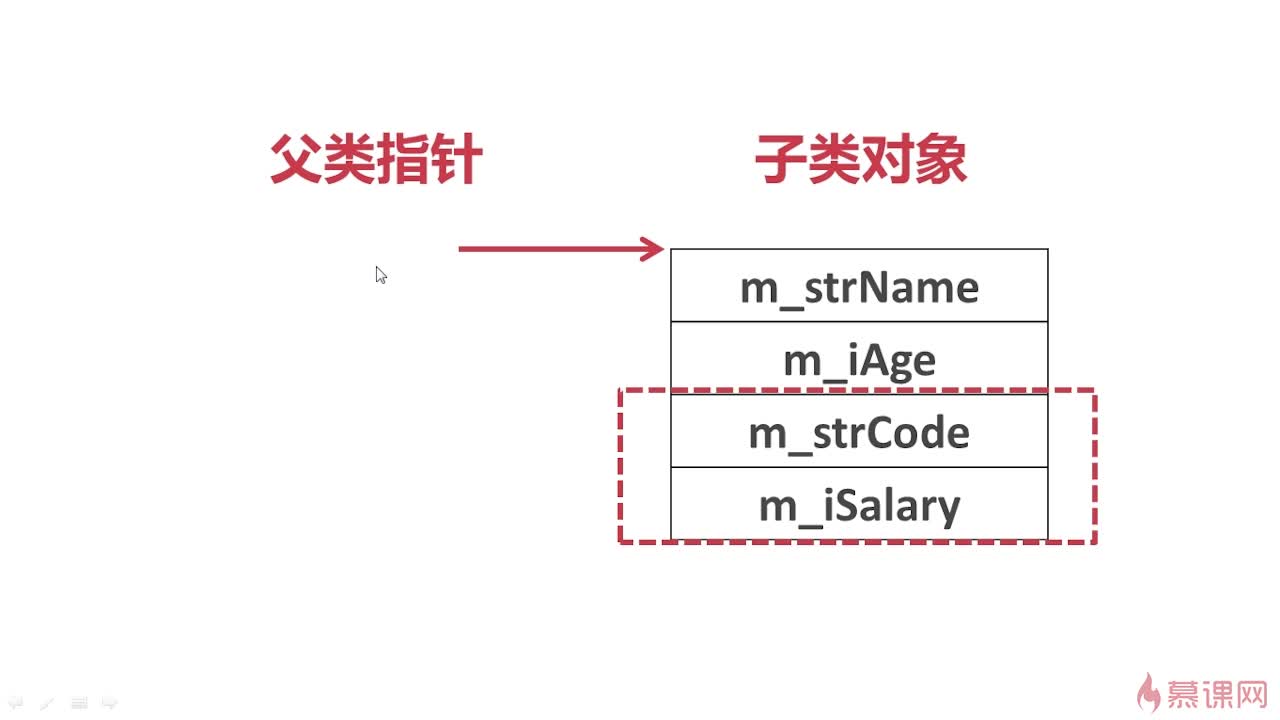
****

**注意：继承时父类和子类的同名函数不会构成重载，只会隐藏即：**

**子类B继承A，若B和A有同名的成员和成员函数，当子类创建的对象调用成员函数时，只会调用子类自己的成员函数，而不会调用父类的成员函数，即便是构成了重载的条件也不行，子类只会调用子类的成员函数，从而隐藏从父类中继承的同名的成员函数，要想访问父类的成员函数要使用A：：function（）**

# 2：is a:

**父类指针只能访问 父类拥有的成员数据**

****

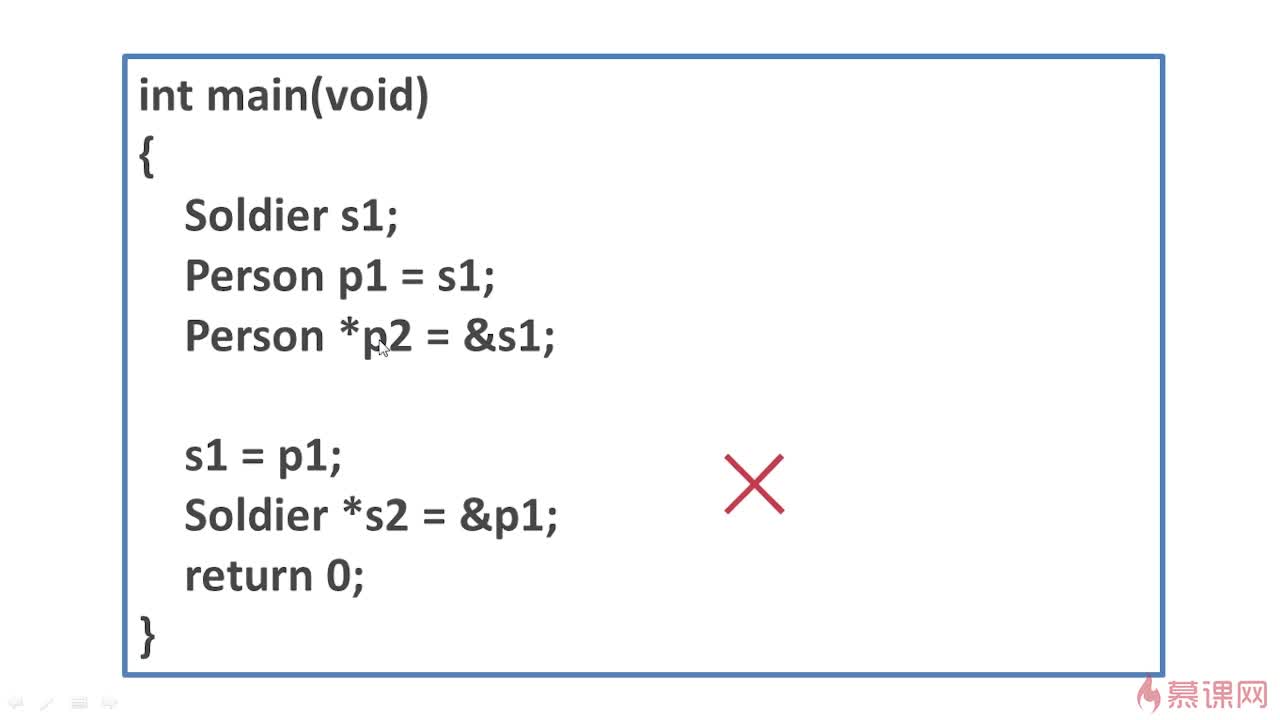
**结论：**

**可以子类给父类赋值，可以让父类类型的指针指向子类，但是：**

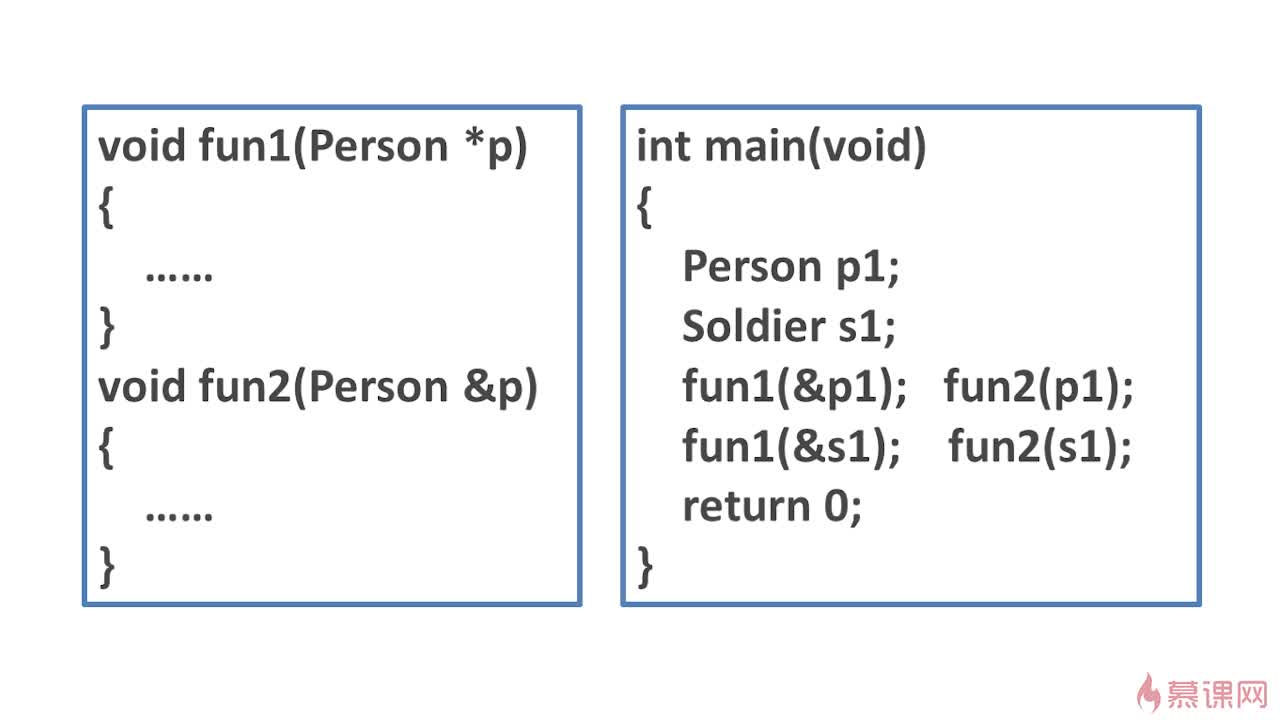
**用子类初始化父类或赋值时，会有子类的数据丢失(只赋继承来的值)**

**用父类指针指向子类的成员函数或数据成员的时候只能指向子类从父类继承来的成员函数或者数据成员。**

**Is a的详解：**

****

**is a 语法 参数的传递：**

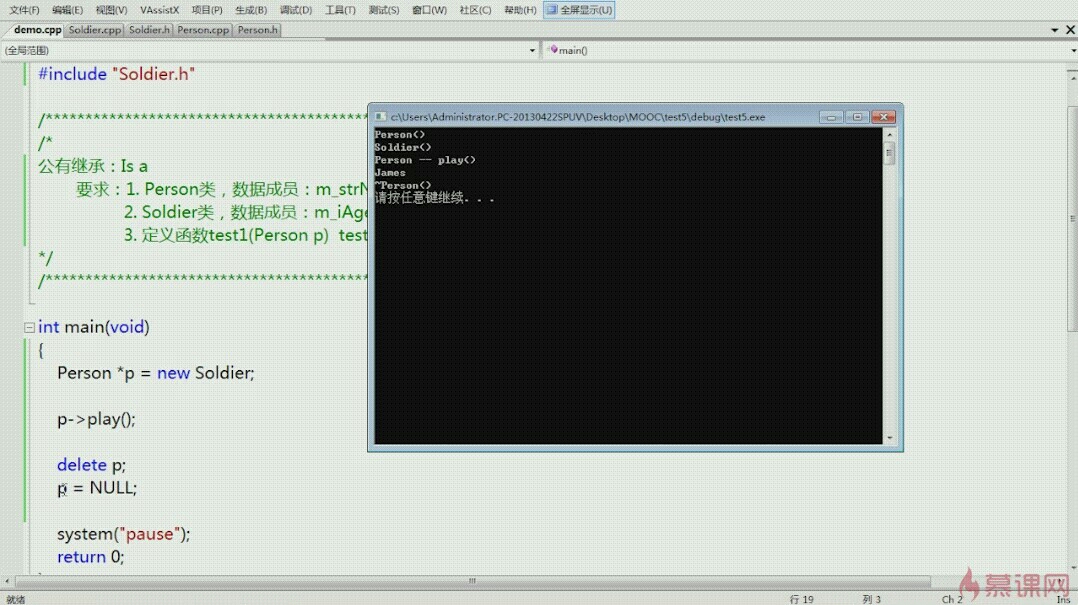
****

## 虚析构函数：

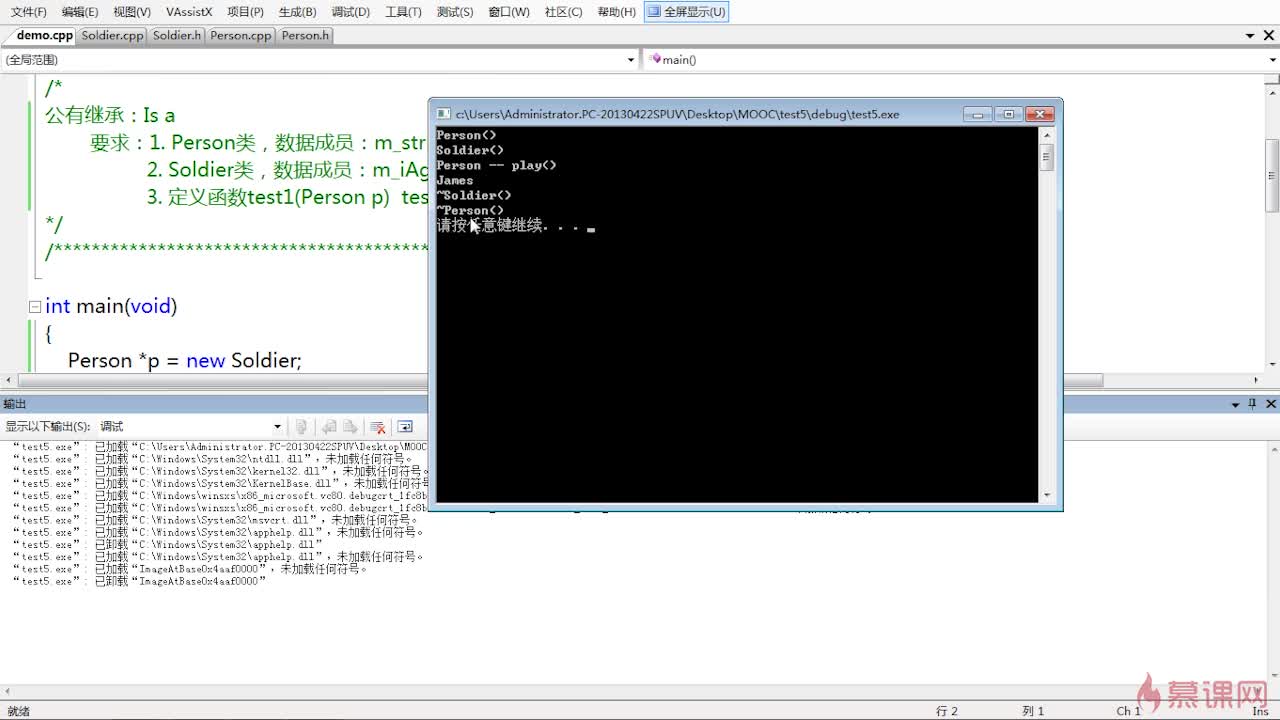
**Is a 中若用父类类型的对象的指针指向子类类型的对象的时候，若在堆中开辟内存（定义子类对象），若不加任何处理，那么最后用delete释放父类的指针指向的内存的时候只能成功地执行父类的析构函数而不会执行子类的析构函数，这样就会造成内存泄漏，所以这时我们就要用到虚析构函数：**

**使用方法：在父类的析构函数前面加上关键字virtual（同时，子类的析构函数前面也可以加上virtual也可以不加virtual，效果时一样的，但是，为了防止不必要的错误，最好在子类的析构函数中也加上virtual）**

**例：不使用virtual时：**

****

**使用virtual后：**

****

**Is a的函数参数传递的几种常见方式：**

**第一种：**

**void test1(Person p)**

**{p.play}**

**分析：调用test1的时候，因为在test1中所定义的参数是一个对象P，所以传值的时候会先实例化临时对象P，通过临时对象P调用play这个函数，在test1执行完毕之后，p这个临时对象就会被销毁**

**如果函数的参数是基类的对象，那么基类的对象和派生类的对象，都可以作为实参传递进去，并且可以正常使用。**

**第二种：**

**void test2(Person &p)**

**{p.play();**

**}**

**分析：test2没有实例化临时对象，使用基类的引用也可以接收基类的对象以及派生类的对象**

**第三种：**

**void test3(Person \*p)**

**{p->play();**

**}**

**分析：**

**的调用结果和void test2(Person &p)一样**

**总结：**

**因为不产生临时变量，所以void test3(Person \*p)、void test2(Person &p)效率更高**

# 3：多继承与多重继承：

**多继承是指一个子类可以继承多个父类（个数不限）；而多重继承是指一个子类继承的父类同时也是另一个父类的子类**

**概述：**

**多重继承：例如有三个类，C类继承了B类，B类继承了A类，那么这三个类之间就存在多重继承关系**

**多继承：一个派生类有多个基类**

**多继承和多重继承**

**多重继承：线性继承两次及其以上为多重继承。例如：**

**class Person:{};**

**class solider:public Person{};**

**class test:public solider{};**

**test 即为多重继承。**

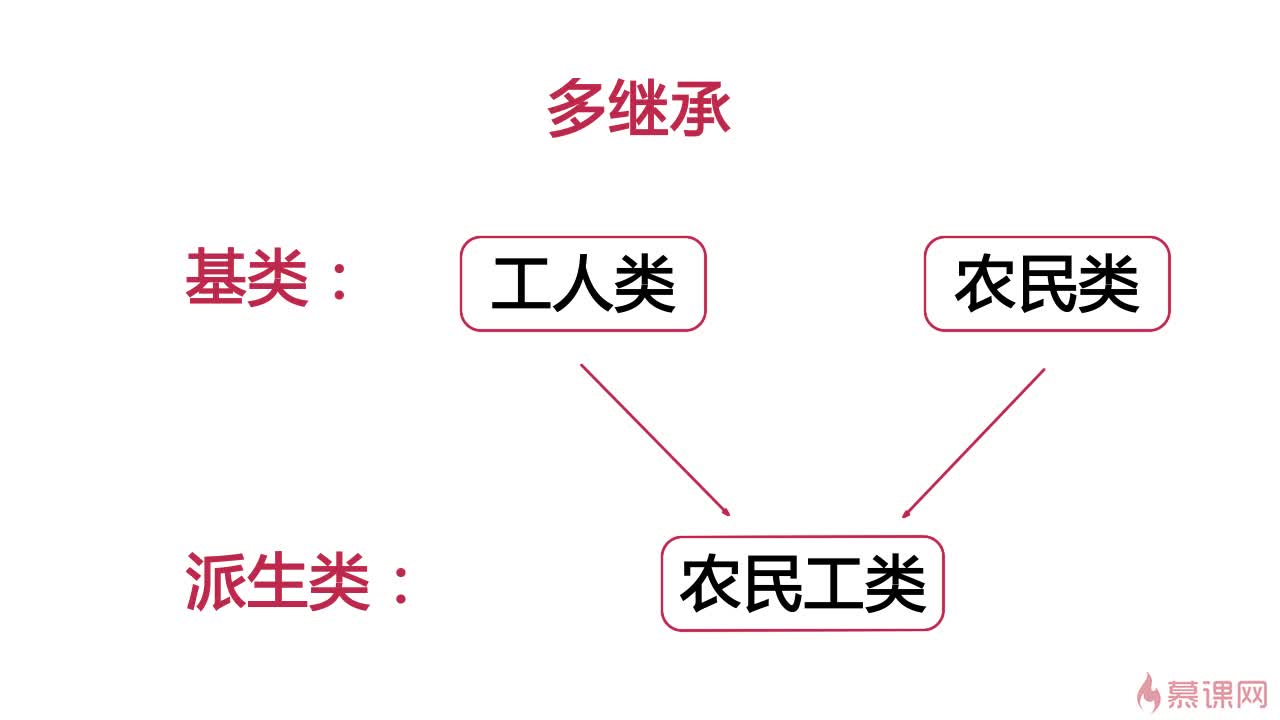
## 多继承：

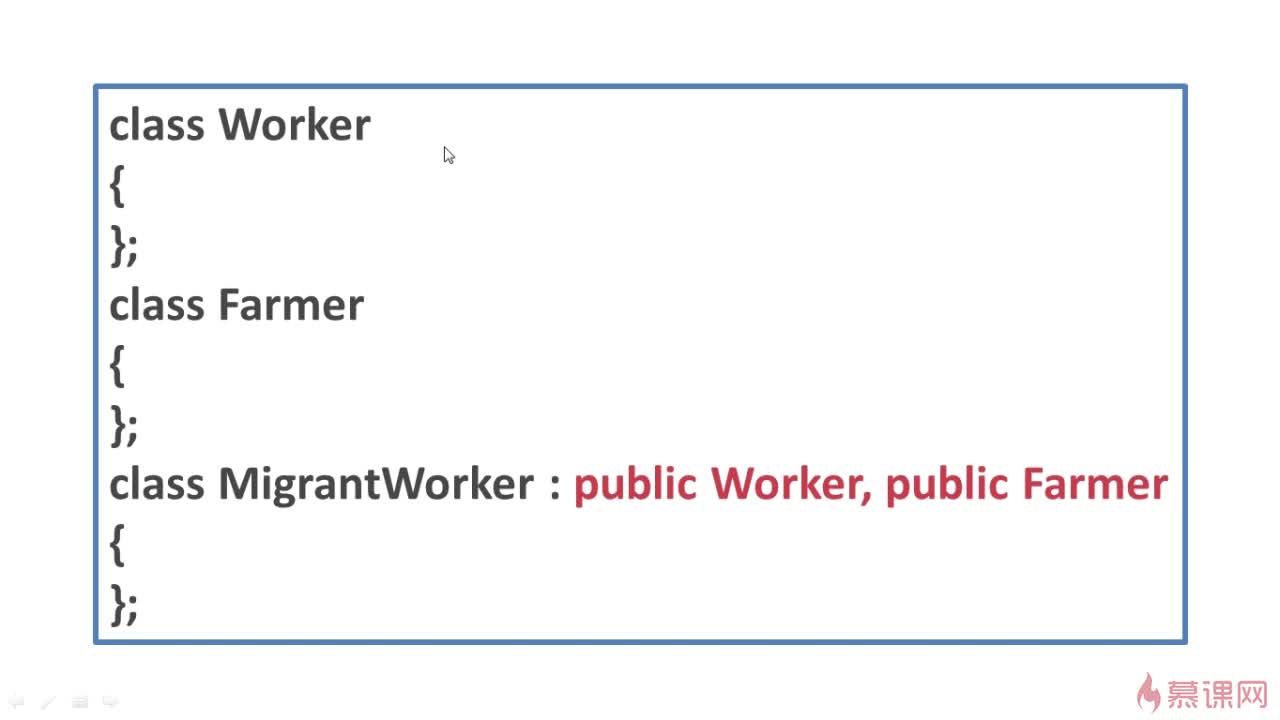
**一次继承连个类成为多继承。例如：**

**class test:public Person,public Solider{};**

**此时test 为多继承。**

**多继承关系图解：**

**多继承代码实现：**

****

**多继承注意点：**

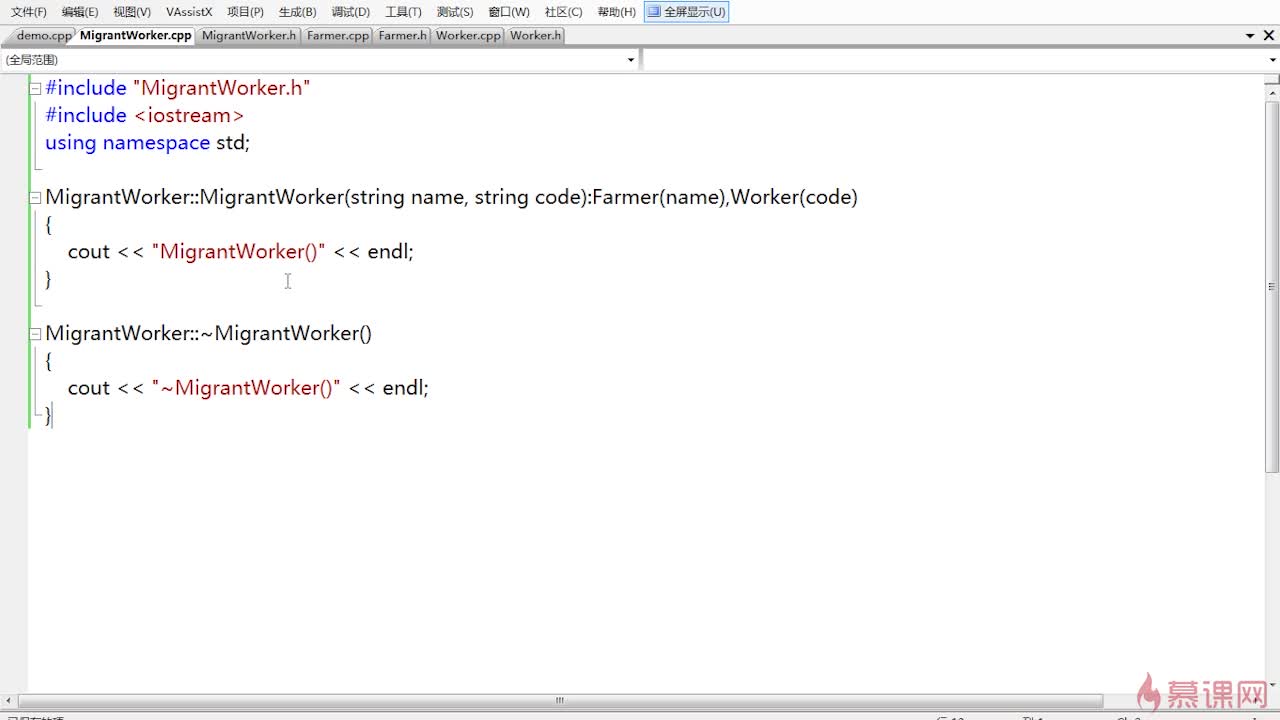
**处于同一层次的各基类构造函数的执行顺序取决于定义派生类时所指定的各基类顺序，与派生类构造函数中所定义的成员初始化列表的各项顺序无关。**

**即定义一个子类的对象时，首先调用父类的[构造函数](https://www.baidu.com/s?wd=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyu-PH9BP19huANbPH7-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EP1c1nj6srjc" \t "_blank)，如果一个子类有多个父类时，则按照声明的顺序依次执行父类的[构造函数](https://www.baidu.com/s?wd=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyu-PH9BP19huANbPH7-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EP1c1nj6srjc" \t "_blank)**

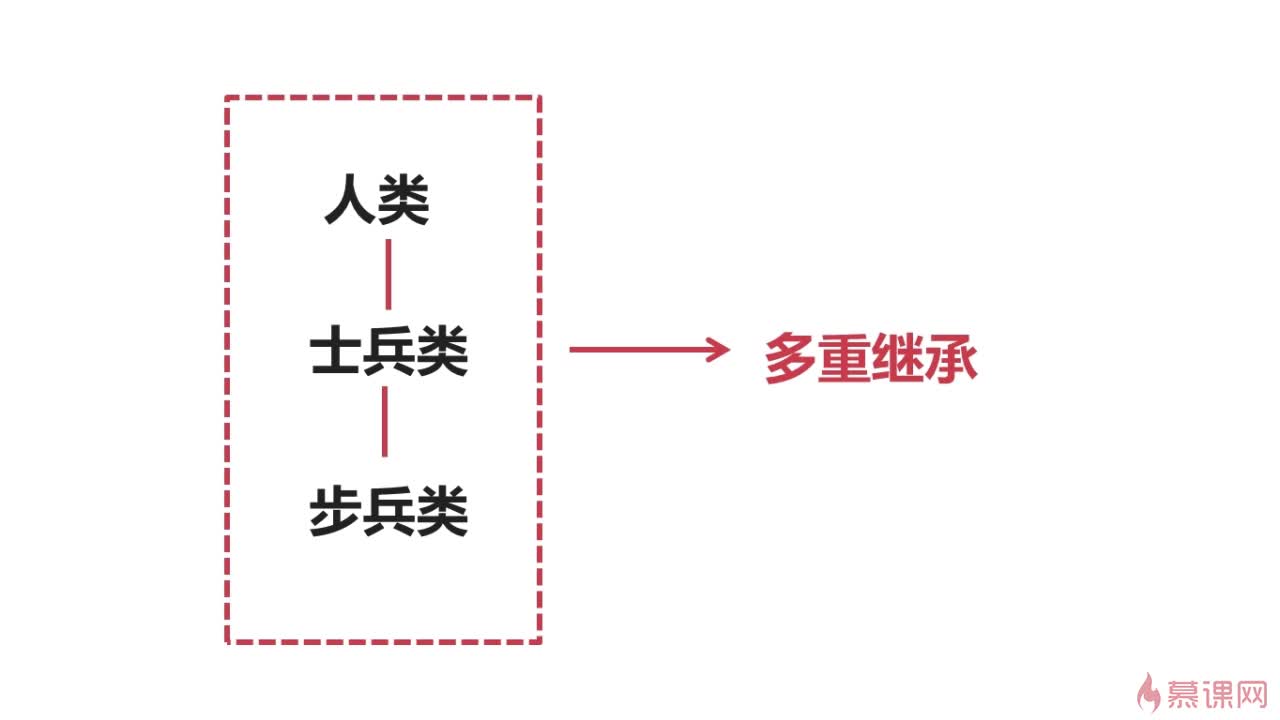
**子类的初始化有两种方法，一是通过调用父类的构造函数实现，二是直接在子类的构造函数中实现。**

**如果要调用父类的带参构造函数，则必须在子类的初始列表中实现；若不想调用的话，则必须在父类当中有默认构造函数；总之就是子类在实例化的时候一定会调用父类的构造函数**

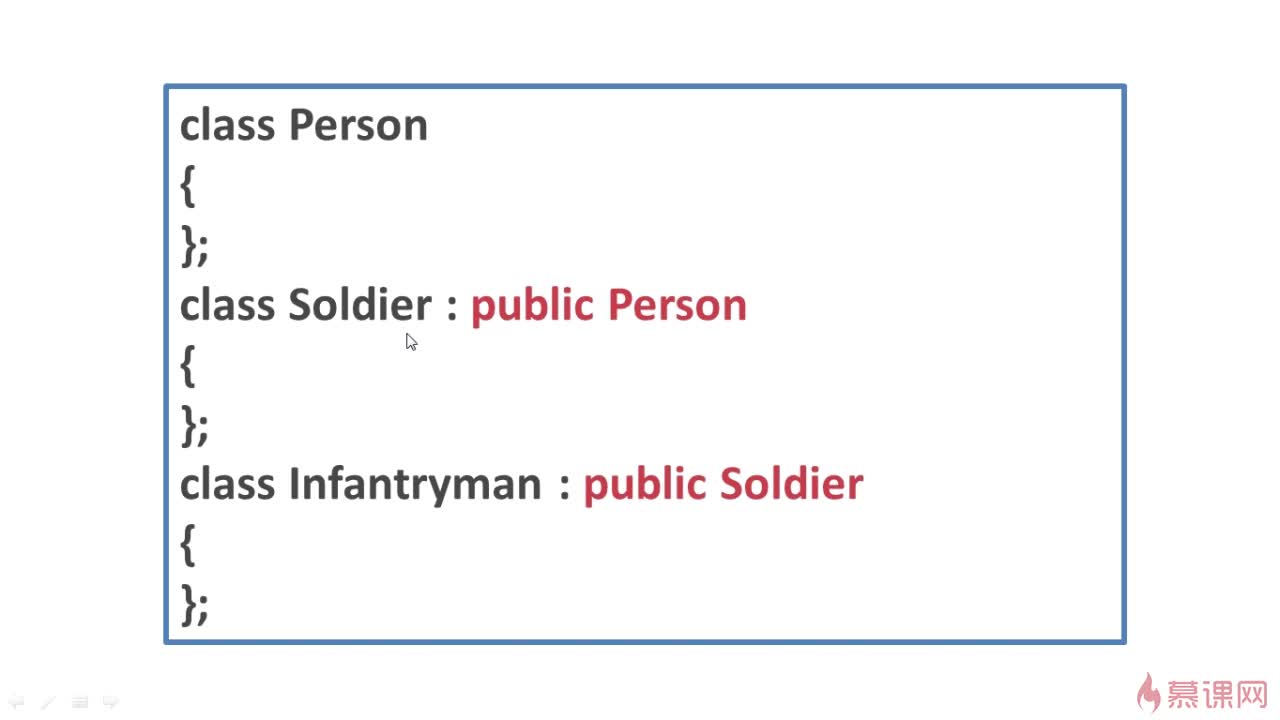
**初始化列表的用法复习：**

****

## 多重继承关系图解：

****

**多重继承代码实现：**

****

**多重继承注意点：**

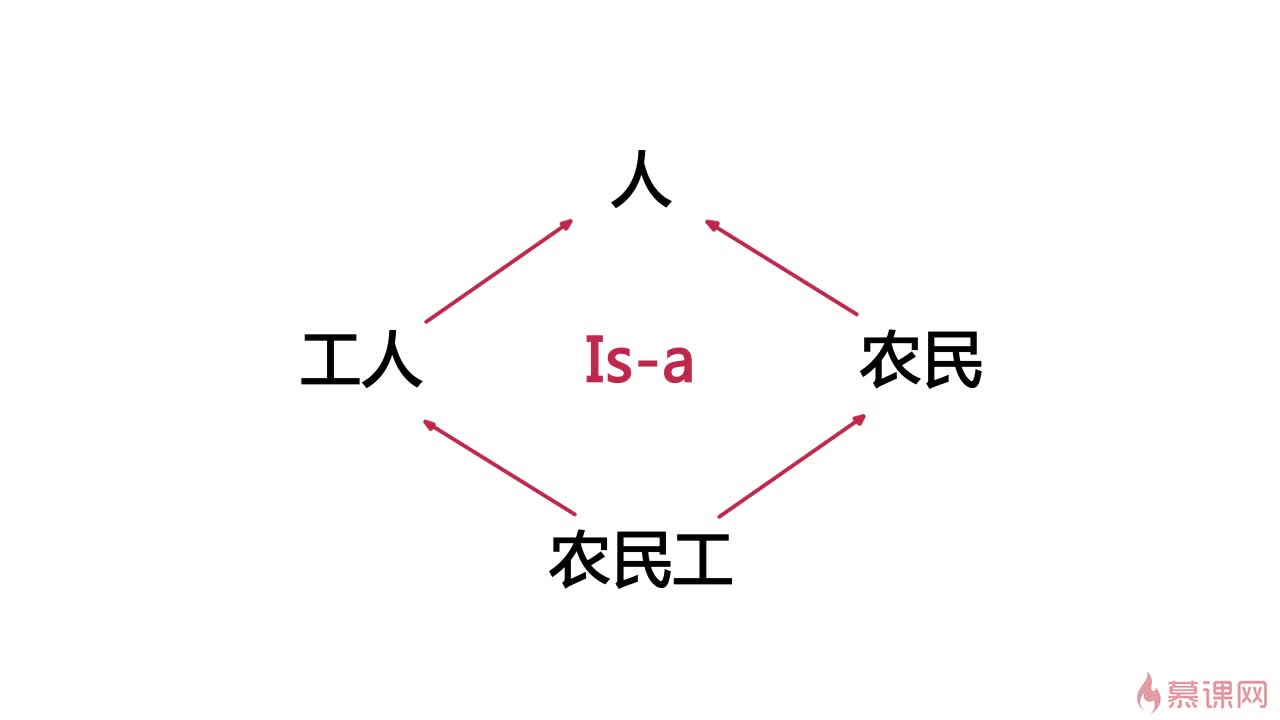
**在最底层的类，要实例化对象，就必须执行被继承量当中的每一个类的构造函数且执行顺序由高到低（即先爷爷再爸爸再自己）**

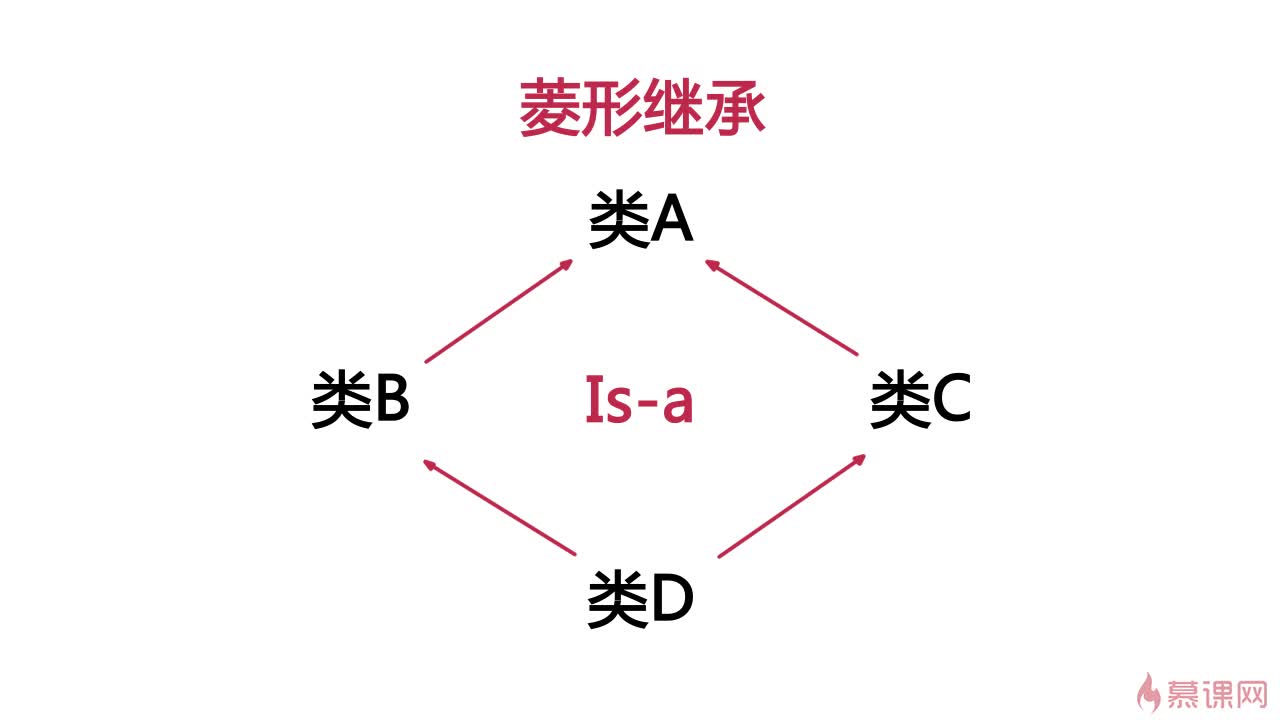
**在销毁的时候，逆序执行**

**无论继承有多少级，只要保持间接或者直接的关系，子类都可与自己的直接父类，或者是间接父类，称之为is a 的关系，并且能够通过父类的指针对直接子类或者是间接子类的对象进行操作**

# 4：虚继承：

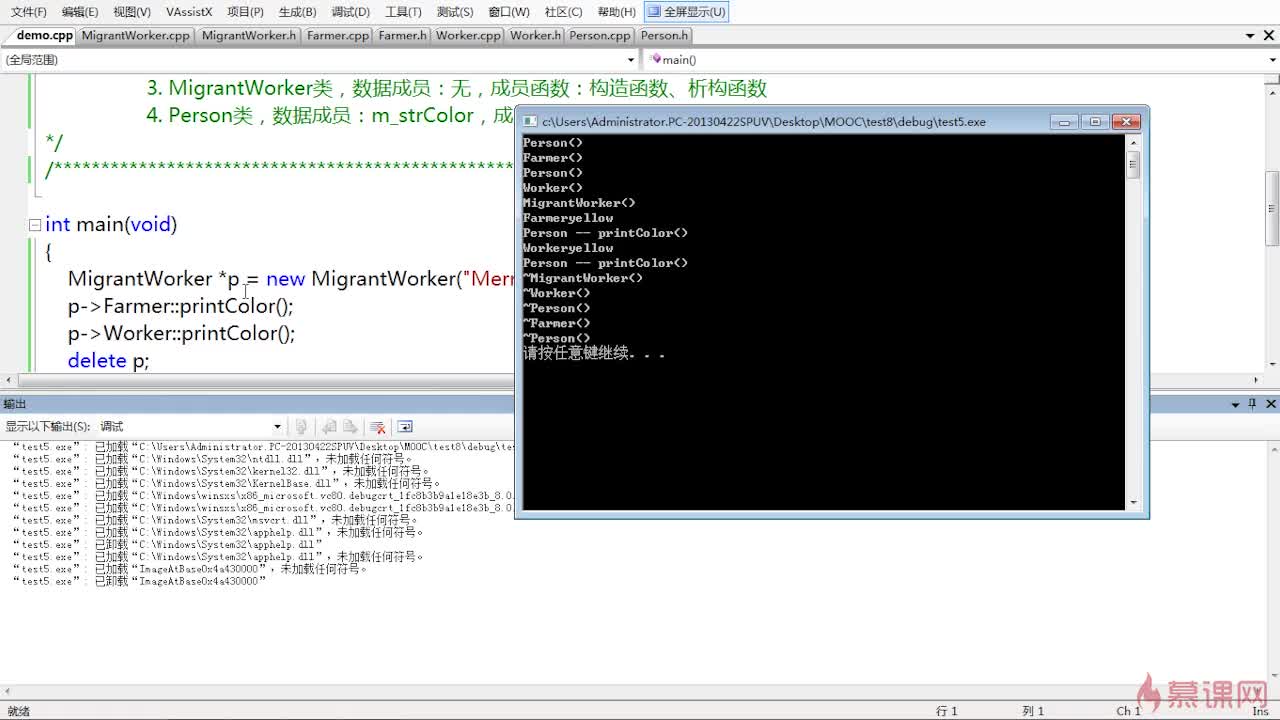
**菱形继承关系图**

****

****

**当发生菱形继承时，在实例化D的时候，会产生两份A的数据（即实例化c和实例化b时产生的两份不同的a数据），形成数据冗余。**

**比如：**

****

**这个实例中我们可以看到农民工的color竟然有两个值（Fameryellow和workeryellow）,这样不就出现问题了吗？所以我们要使用虚继承来解决这个问题：**

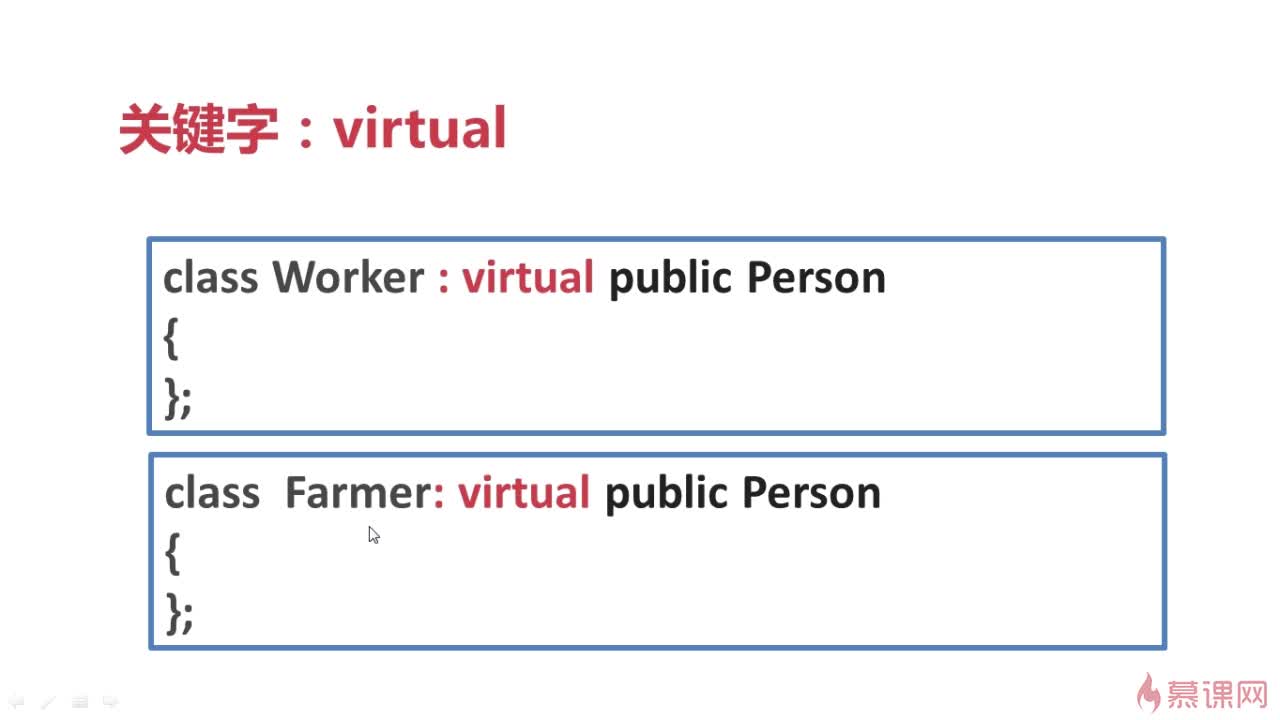
**例如.class B : virtual public A{};**

**Class C : virtual public B {};**

**那么D在继承B和C的时候即可直接继承。**

**Class D : public B, public C {};**

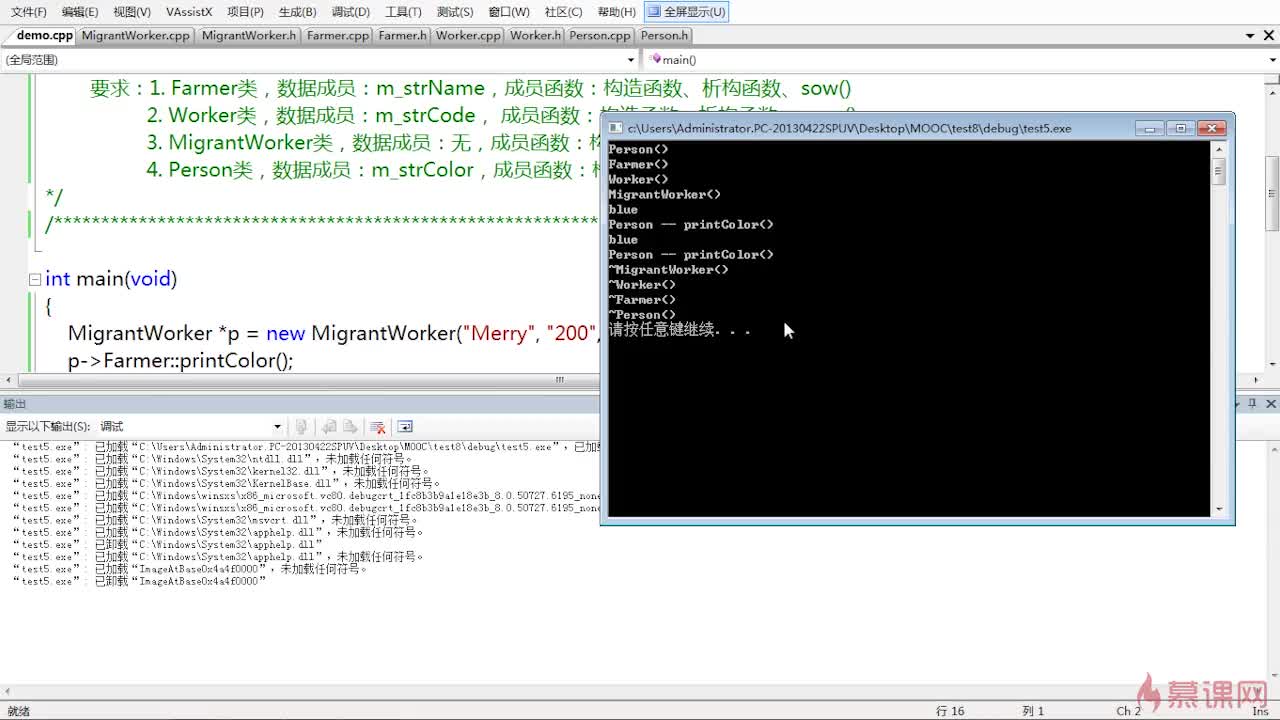
**虚继承的具体格式：**

****

**需要注意的是，以上写法也可写成class Worker : public virtual Person{};**

**也就是说，virtual 关键字即可写在public前面，也可写在public后面，但是一般来说建议写在public前面。**

**同样是上面的那个例子，在worker和Famer引用person的时候分别加上virtual关键字后出现了以下效果：**

****

**可以发现，虚继承父类会使用默认构造参数，不接受子类传入的数据！！！！**

**即使用虚继承之后作为菱形继承最顶层的父类，不会再进行参数的传递，参数只使用了顶层父类的默认参数，而无法从子类中获得传入的参数**

**首先virtual词典里解释为本质上存在但是没有表现形式，而C++中 注意 virtual 写在了共有继承前面，意思就是在继承时是真实存在但是不能被访问；换句话说，你在实例化对象MigrantWorker时需要实例化Farmar和Worker，而实例化这两个对象需要先实例化一个Person，那么传入值你选谁的？所以系统此时设置为禁止访问，在对象生成后你可以随便访问，因为你不可能通过两种方式同时访问一个对象内的成员。采用初始化列表的方式来赋值，Per实例化时是不能被访问的因此值传不进去，如果你Far和Wor的构造函数体内用 m\_strColor=color是可以改的。因为执行有先后顺序，不冲突**

# 5：宏定义的应用（作用）：

**宏定义可以用来解决重定义问题，那么什么是重定义呢？**

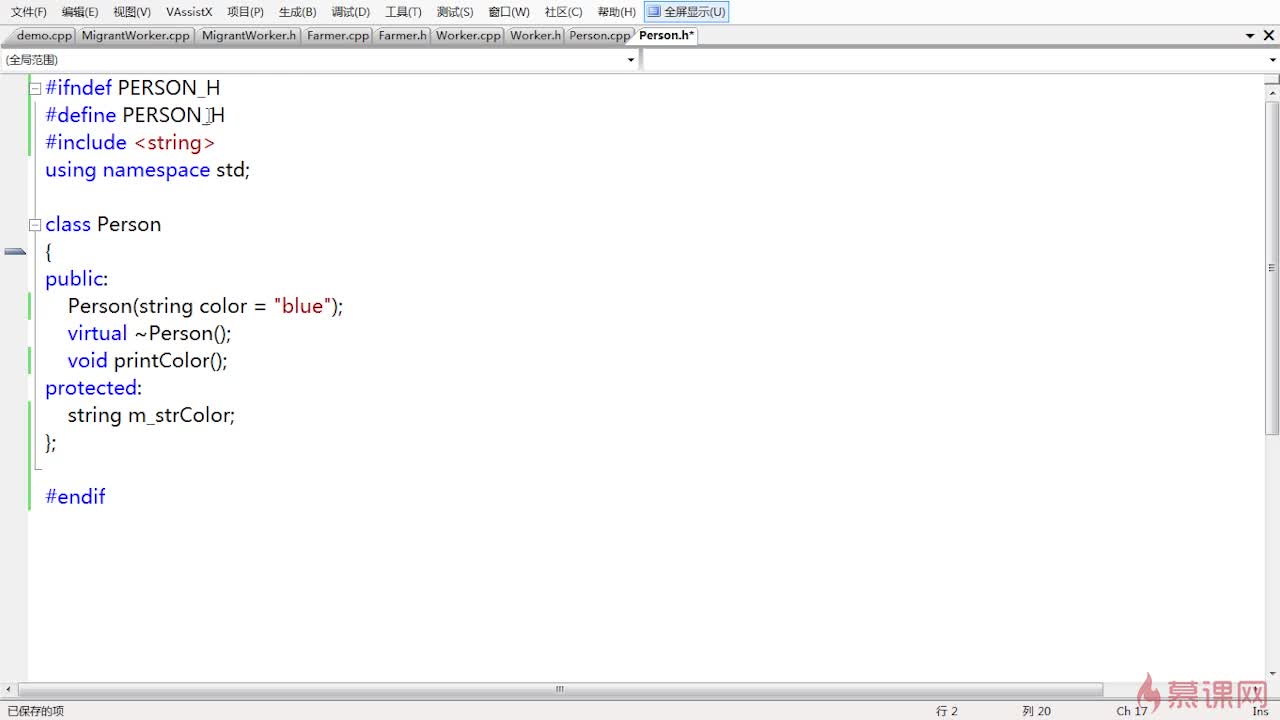
**重定义：**

**比如现在有三个类：person、worker、MigrantWorker，想在worker、MigrantWorker中调用person，那么就会在调用的时候在worker和MigrantWorker的.h文件中分别写上：**

**Class worker:public person{}**

**Class MigrantWorker:public person{}**

**这样在一个以上的类中都写上了对person的定义，系统在编译的时候就会提示“重定义”，为了解决这种重定义的问题，我们使用宏定义：**

****

**菱形继承会导致爷爷重定义的问题<br>**

**解决方法：宏定义（在被重定义的文件里实现，最好其他文件也都写）**

**#ifndef ...//这里写什么都可以，但是建议写的和被重定义的类一样的名字，便于区分**

**#define ... //这里写什么都可以，但是建议写的和被重定义的类一样的名字，便于区分**

**...**

**...**

**#endif**