**快捷键：**

**注释掉选定的代码 control +k +c**

**取消 注释 快捷键 ctrl + k + u**

# 对象成员与对象数组

## 1：对象数组

**堆中实例化对象数组：**

**Coordinate \*p = new Coordinate[3];（分配三次内存，调用三次构造函数）**

**p[0].m\_iY = 20; （数组的第一个成员）**

**p->m\_iY = 20;（数组p）**

**delete []p; //销毁三次，调用三次析构函数**

**p = NULL;**

**return 0;**

**栈中实例化对象数组：**

**Coordinate coord[3];**

**Coord[0]=20;**

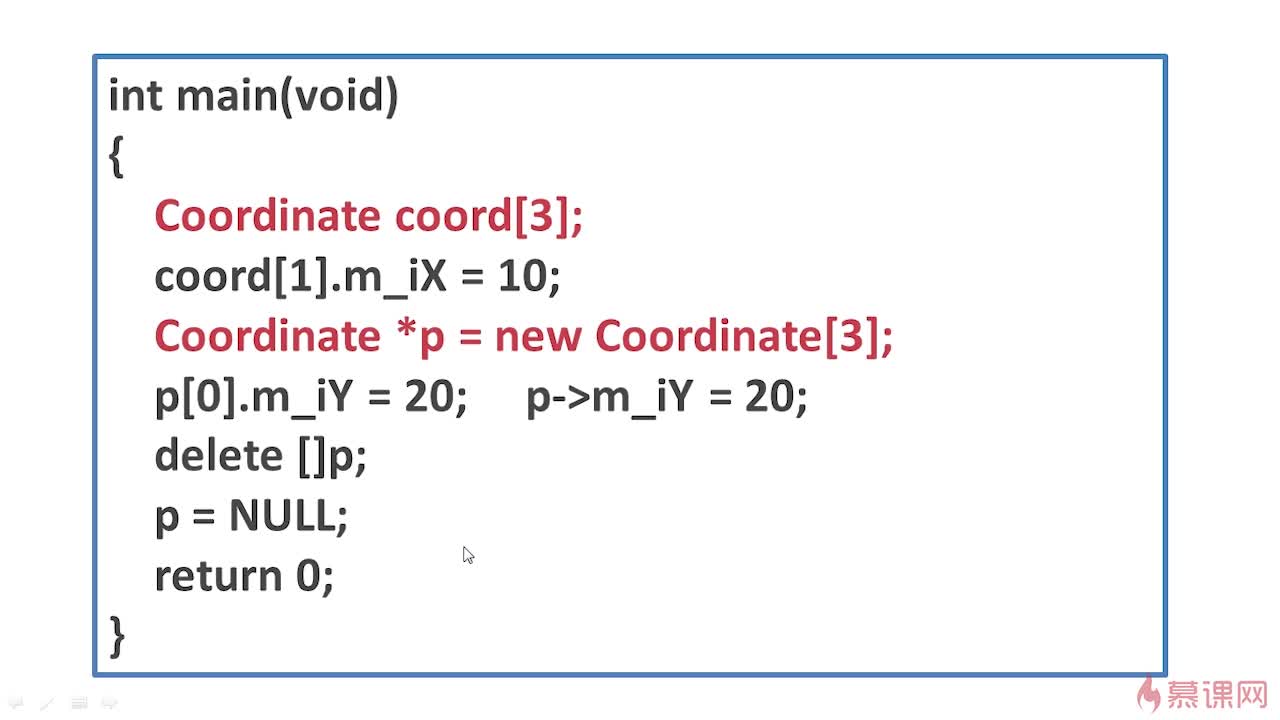
**两种方式都可以**

**分配内存的时候**

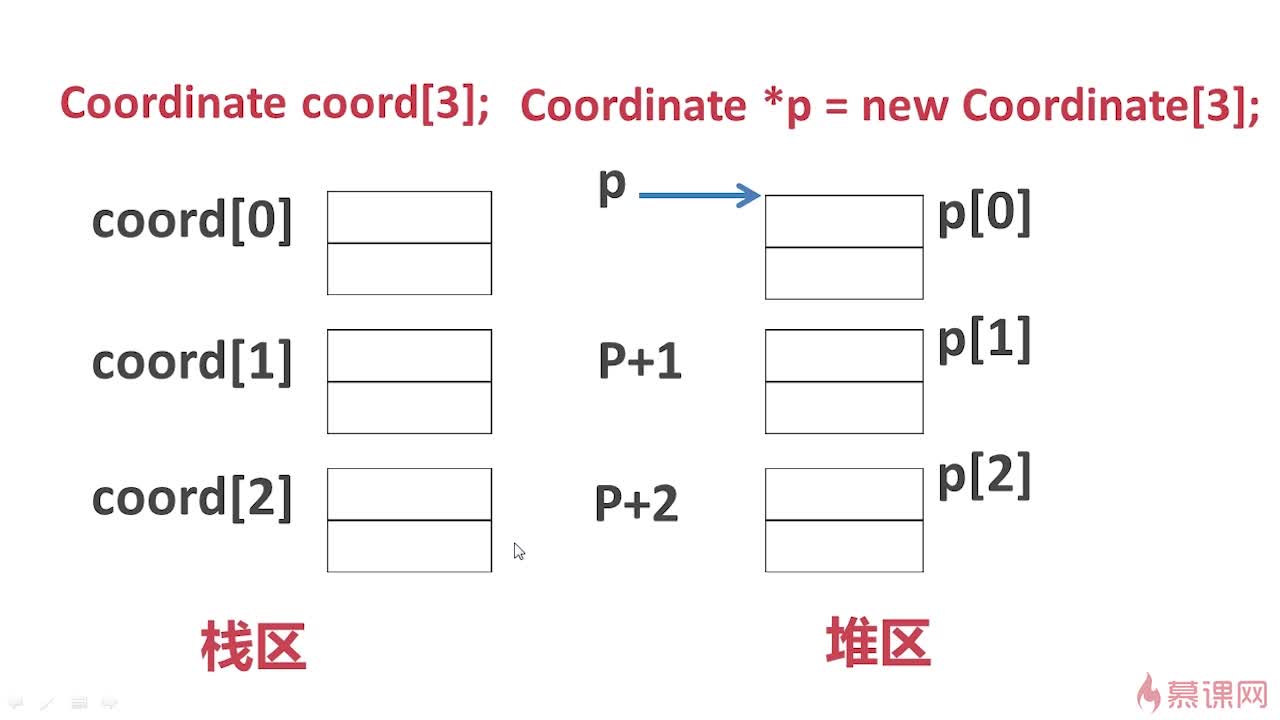
**heap:p向下指 p+1 p+2..**

**stack:最晚放的最先取出**

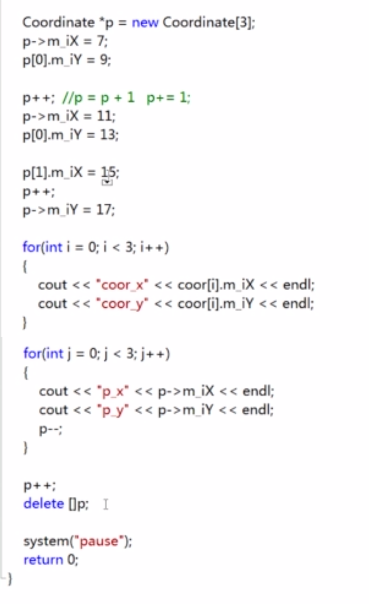
实例：



栈区和堆区定以对象数组的内存分配方式：



注意下面这个例子：



**在上面一个例子中应该特别注意p指针指向内存的位置，特别是最后delete的时候，一定要保证p指向刚开始的位置，即delete []p释放内存时，要确定此时p指的是申请时的首地点，最后p=NULL;防止p成为野指针**

**，否则p指向未知的位置，释放的内存也将会是未知的，后果很严重！！！**

**对delete []p:**

**加上[]能销毁所有的元素，如果不加只销毁第一个元素**

**其实就是调用了p中的析构函数**

**写入delete来释放堆里的内存 使得对象调用自己的析构函数来释放内存。**

**而栈中的对象系统会自己回收内存。**

## 2：对象成员

**定义：一个对象作为另一个对象的数据成员，称为“对象成员”**

**对象成员的创建次序：先创建成员对象，再创建包含着成员对象的对象。**

**对象成员的销毁次序：先销毁包含了成员对象的对象，再销毁成员对象。**

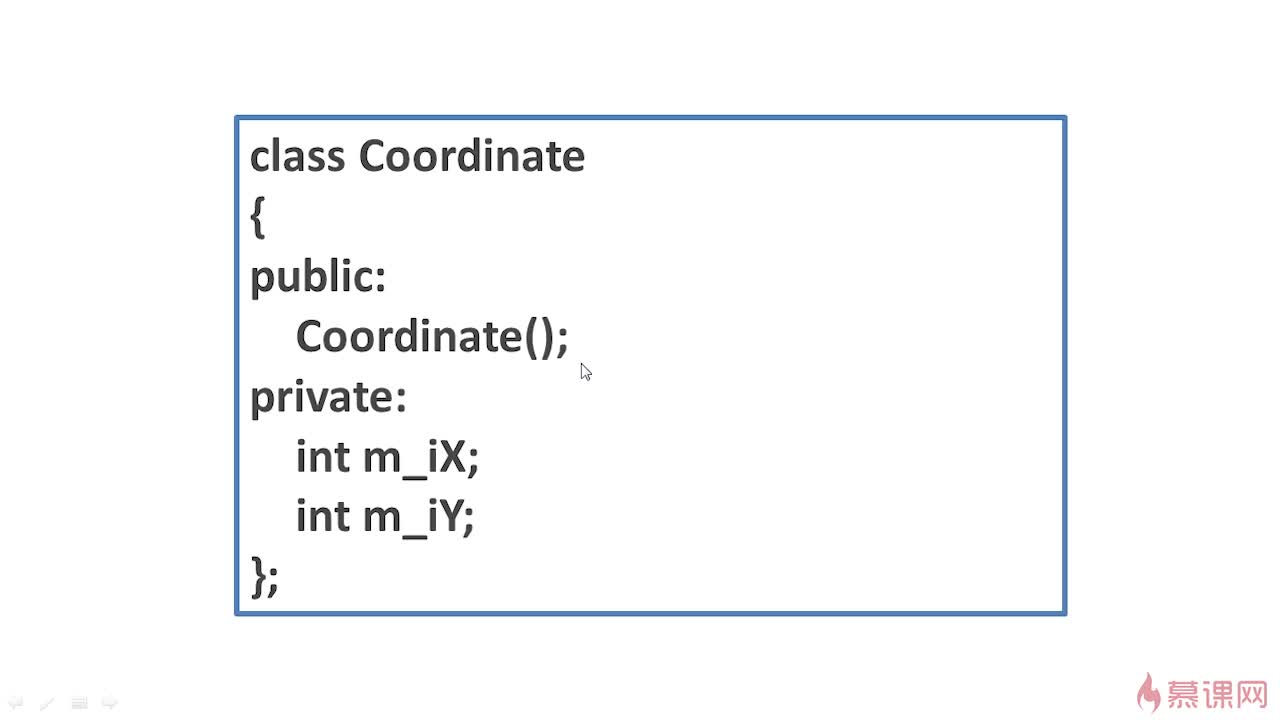
**当没有自己定义的构造函数和析构函数的时候，系统都会自动生成。但是系统生成的构造函数和析构函数是空函数，没有实际作用。定义构造函数可以发给类成员设置默认值等等，同样的，如果我们的类数据成员包含了从堆中申请的内存，类的析构函数就应当加入包含释放堆中内存的语句。或者我们需要程序在对象解散时在屏幕上输出提示信息，也可以在类的析构函数中加入提示信息。**

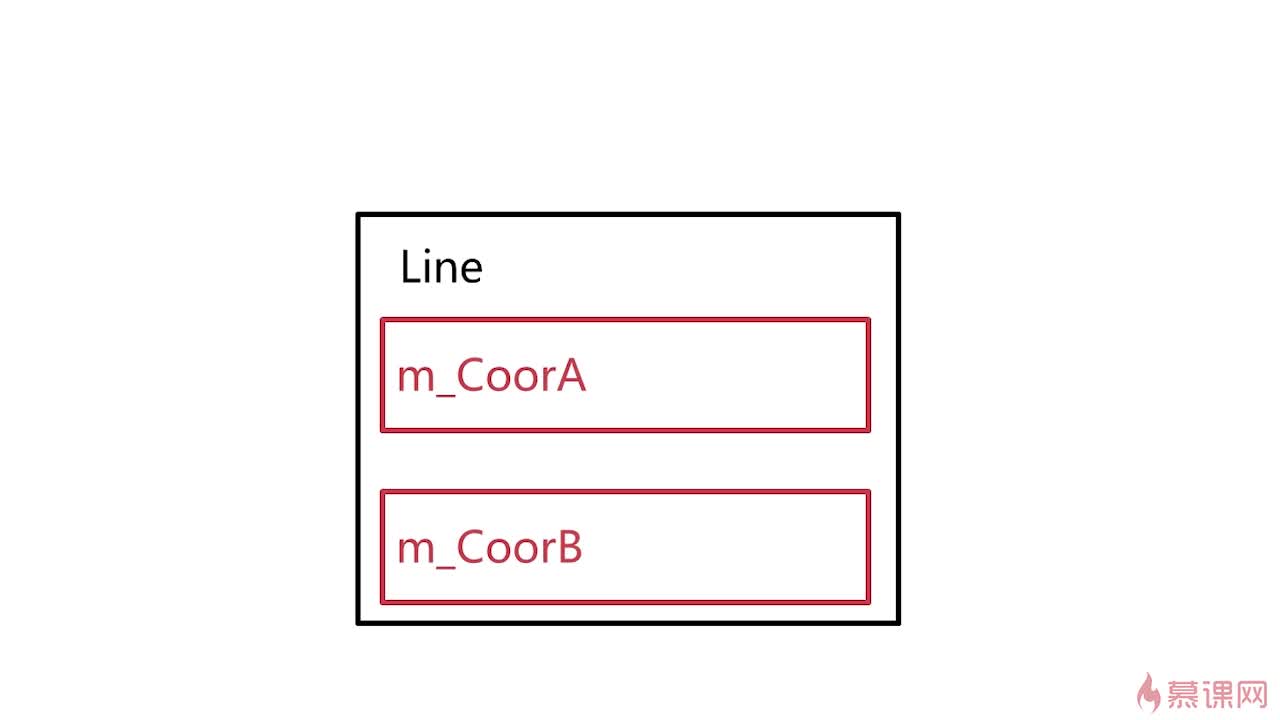
**例：**

**对象成员初始化和销毁时的顺序：**

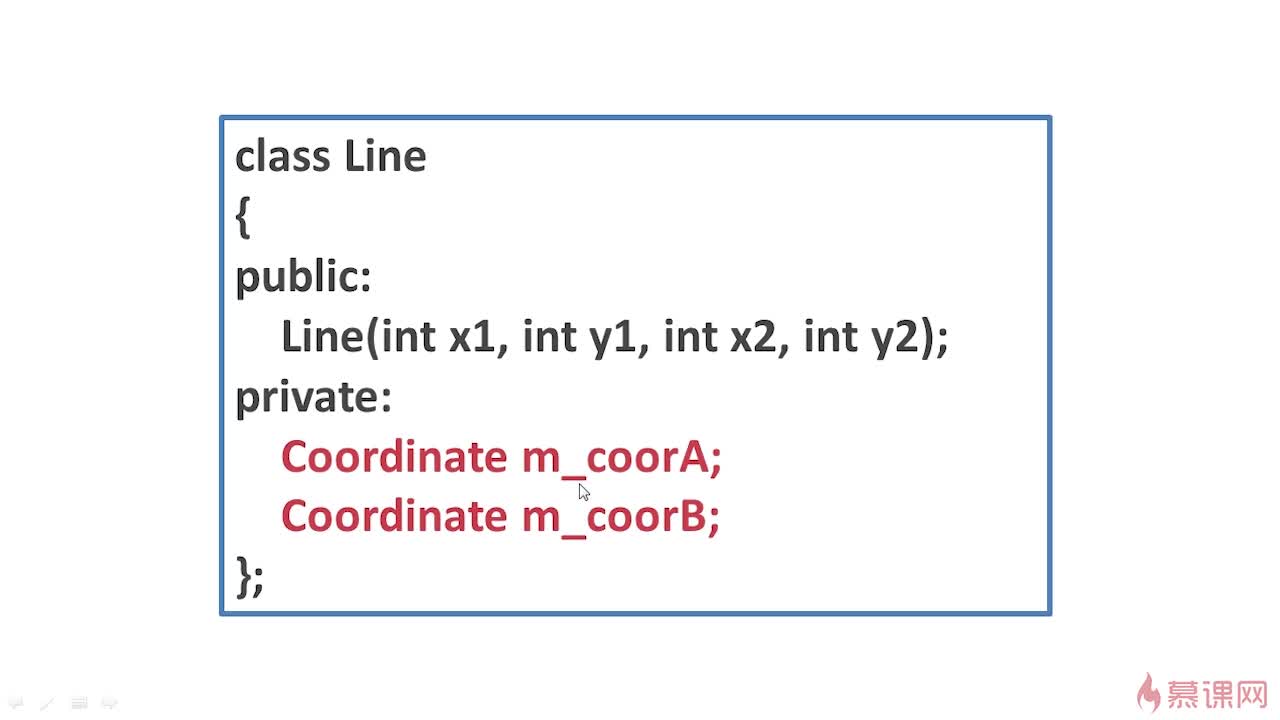
**初始化时：m\_CoorA,m-CoorB,line**

**销毁时的顺序和初始化时的顺序相反：line，m-CoorB，m\_CoorA**



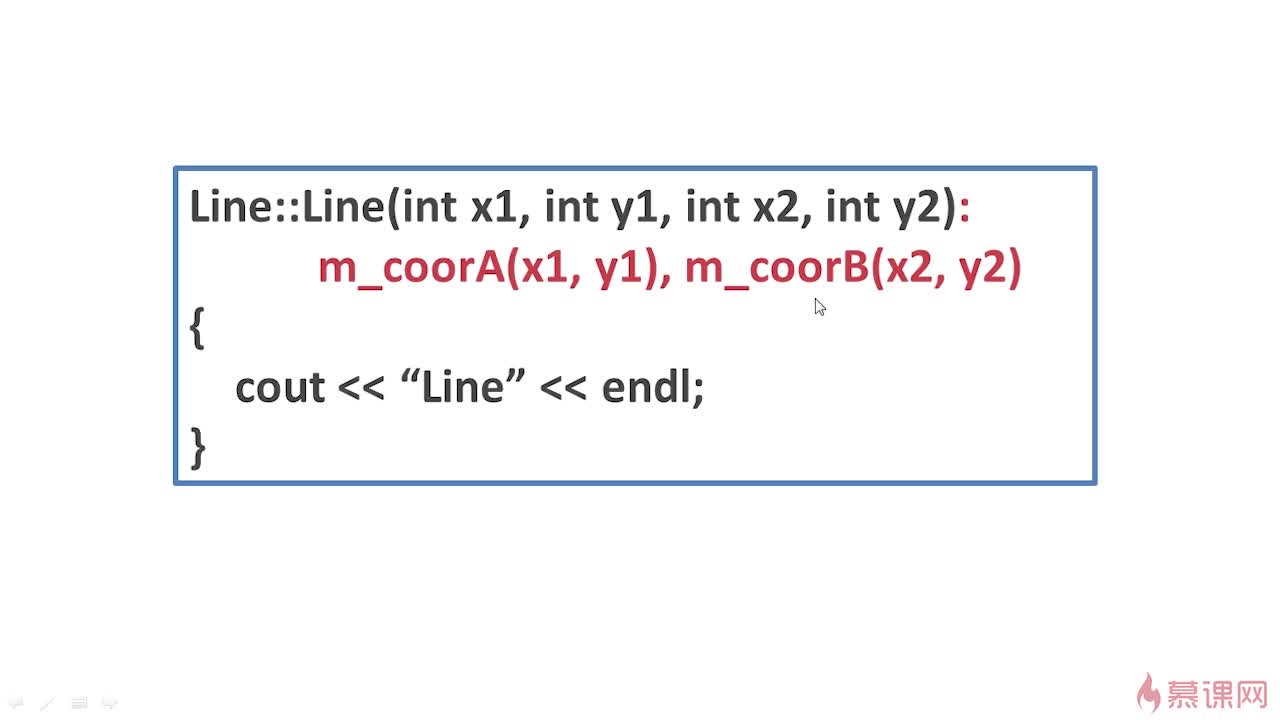


**含有其他对象的对象的构造函数的参数的传递**

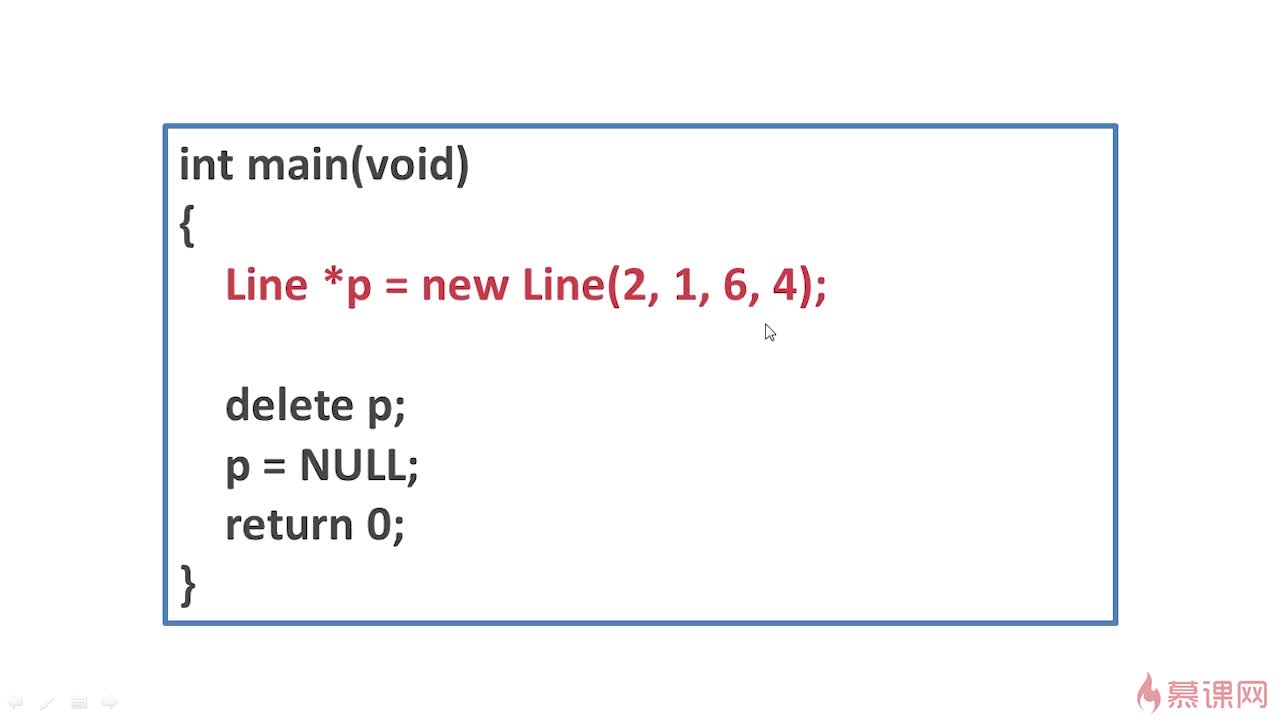


**改进：**

**需要在构造函数中建立初始化列表（两个数据成员），把参数传入到两个数据成员的构造函数中去。**

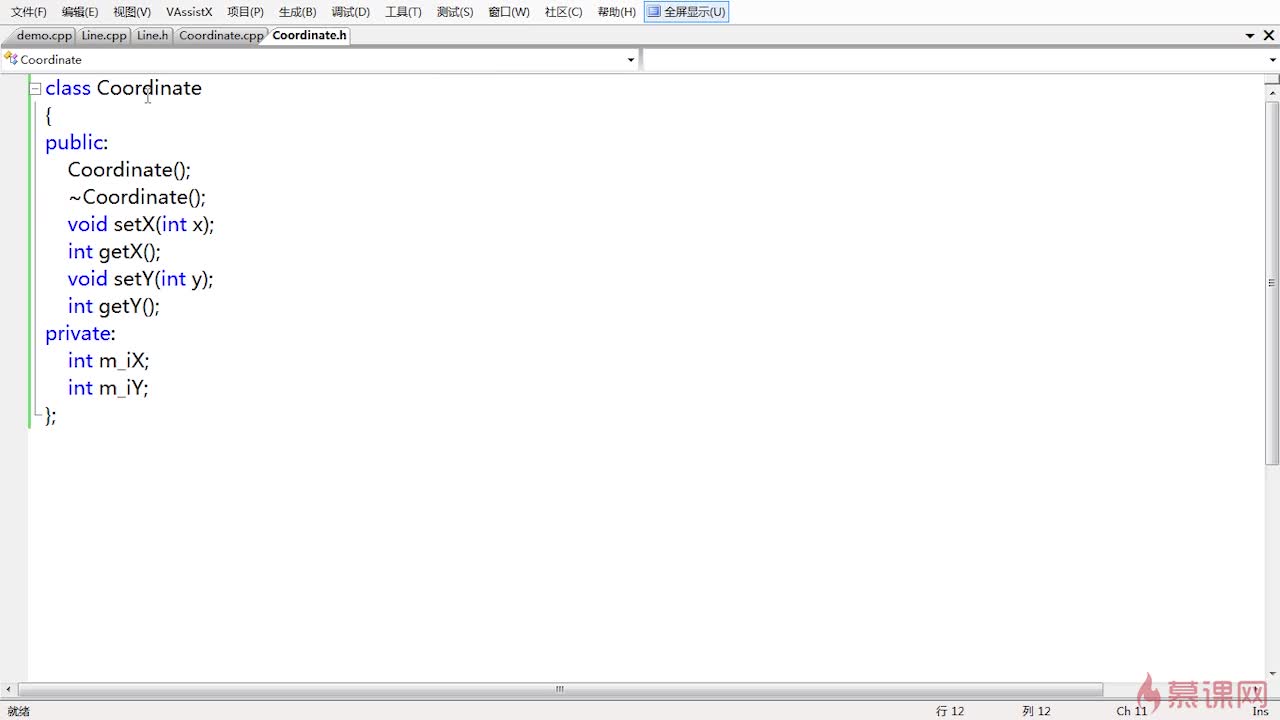


**调用的时候：**

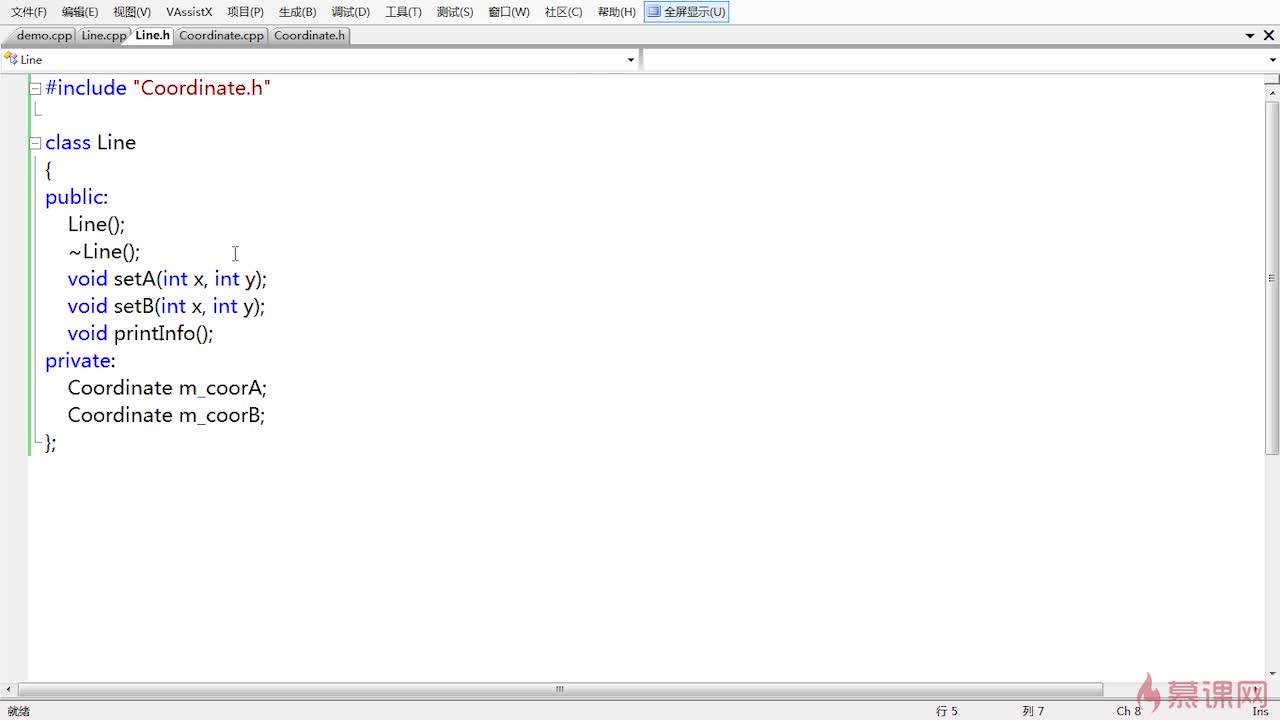


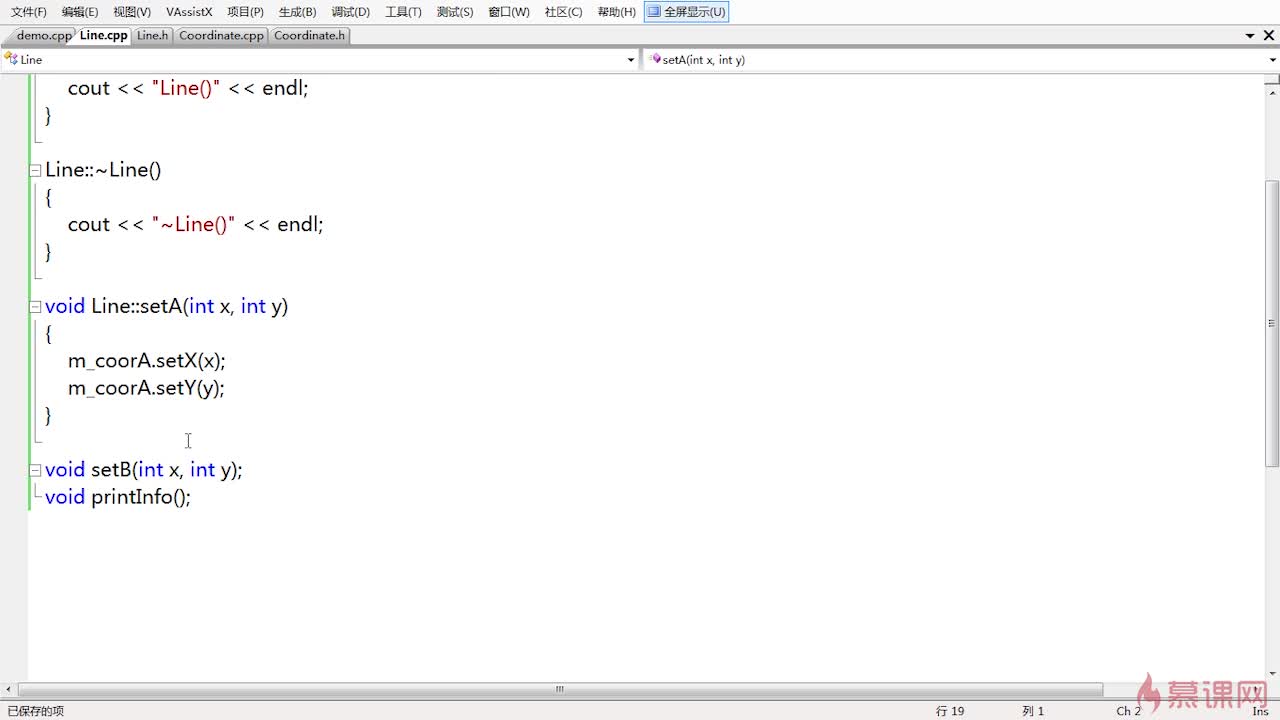
**实例：**

**（1）：坐标类的声明(line.h)：**

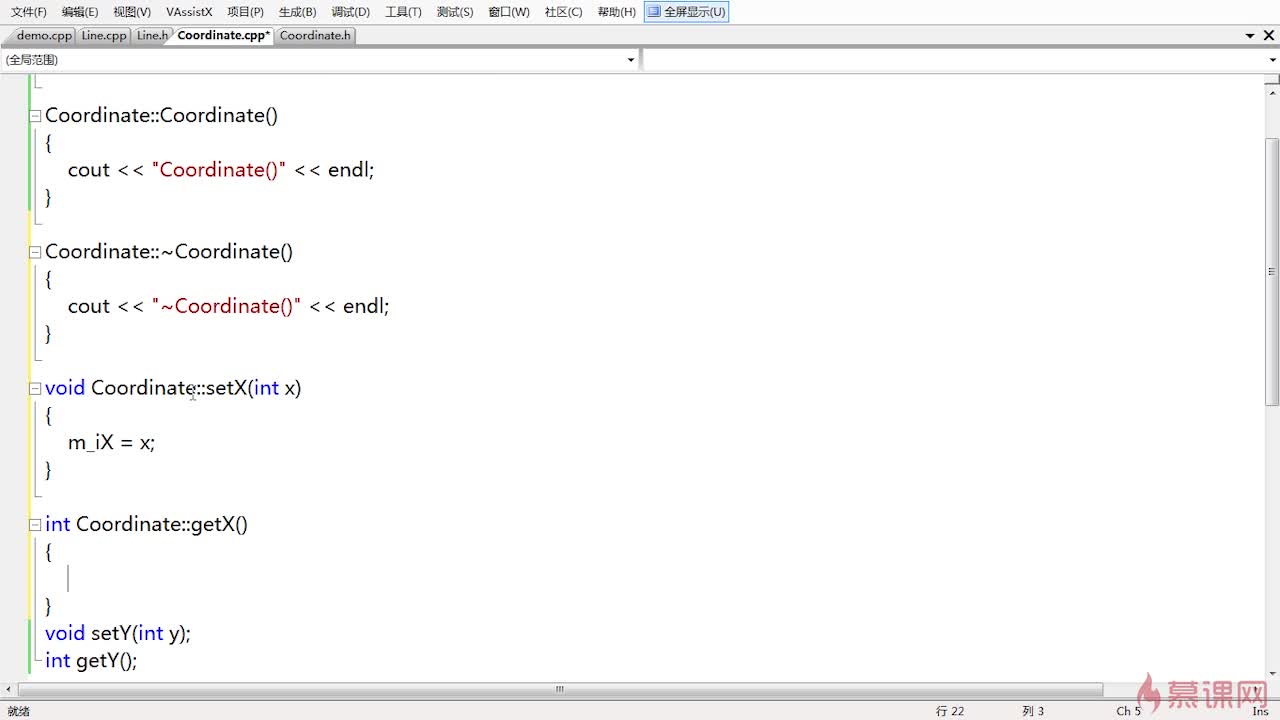


**（2）：线段的声明(line.h)：**



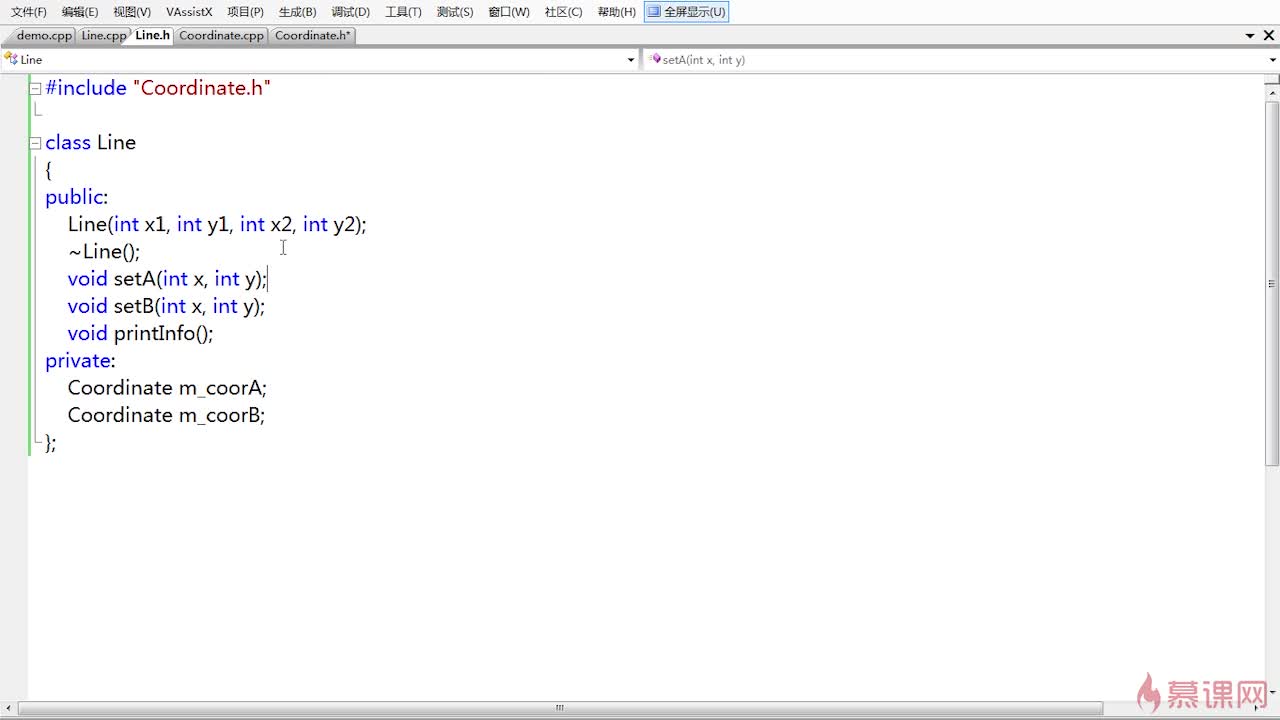
**（3）：线段类的定义(line.cpp)：**

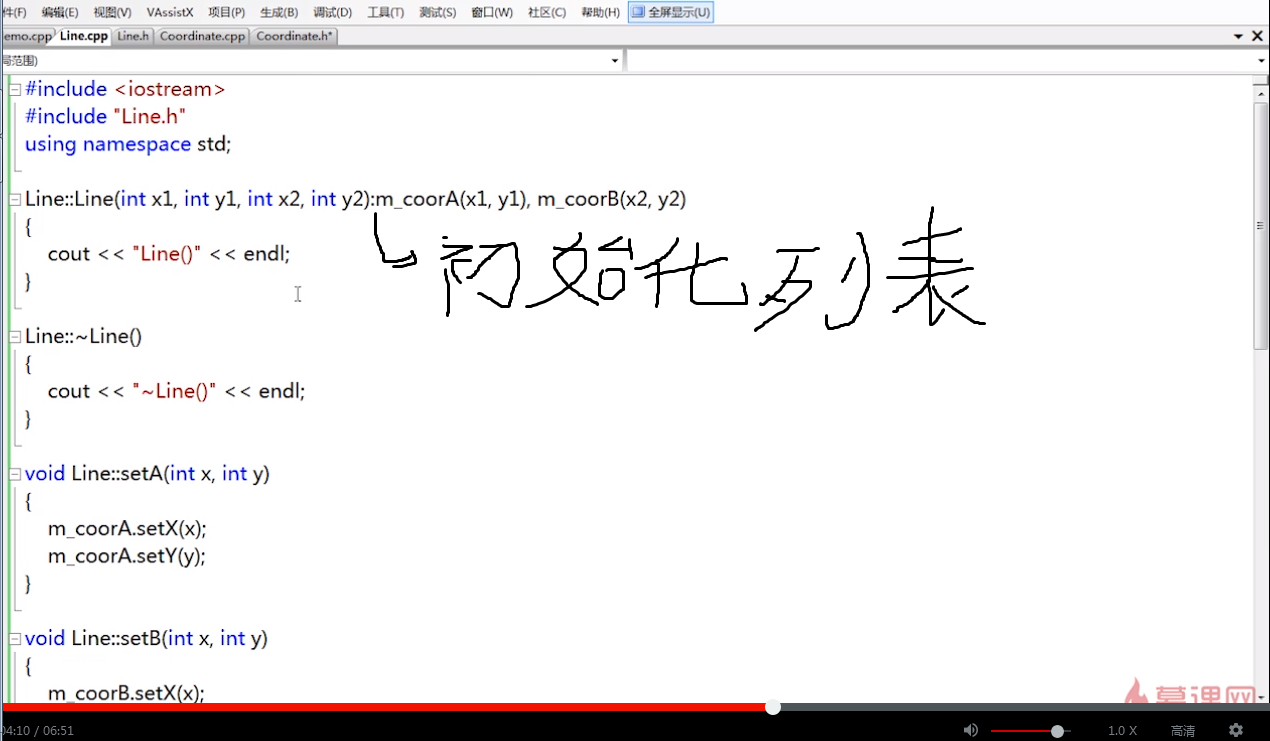
**（4）：坐标类的定义（coordinate.cpp）：**

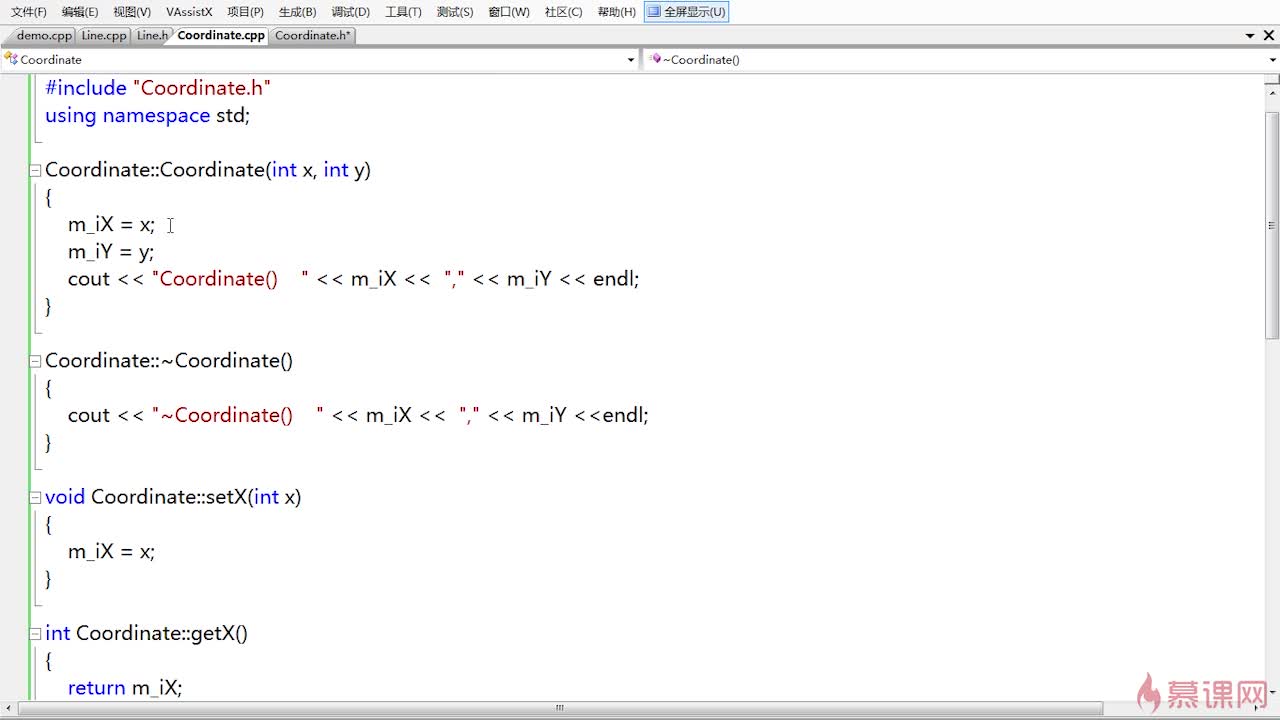


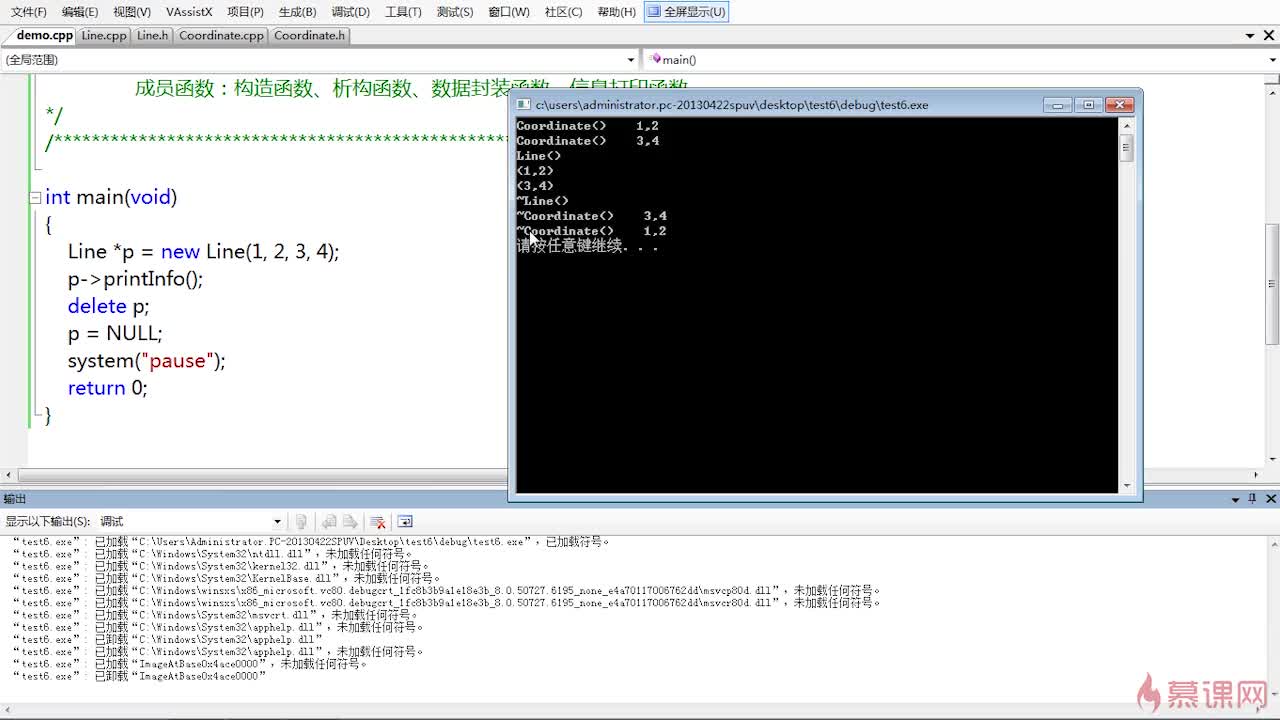
**实例化Line时，要先实例化m\_coorA和m\_coorB,但是这两个对象没有默认构造函数，所以要用初始化列表。（初始化列表执行 在构造函数之前！！！）**

**如果Coordimate类有默认构造函数，也就是不带参数的构造函数，那么Line类就可以不使用初始化列表。如果只有带参数的构造函数，那么必须用初始化列表**









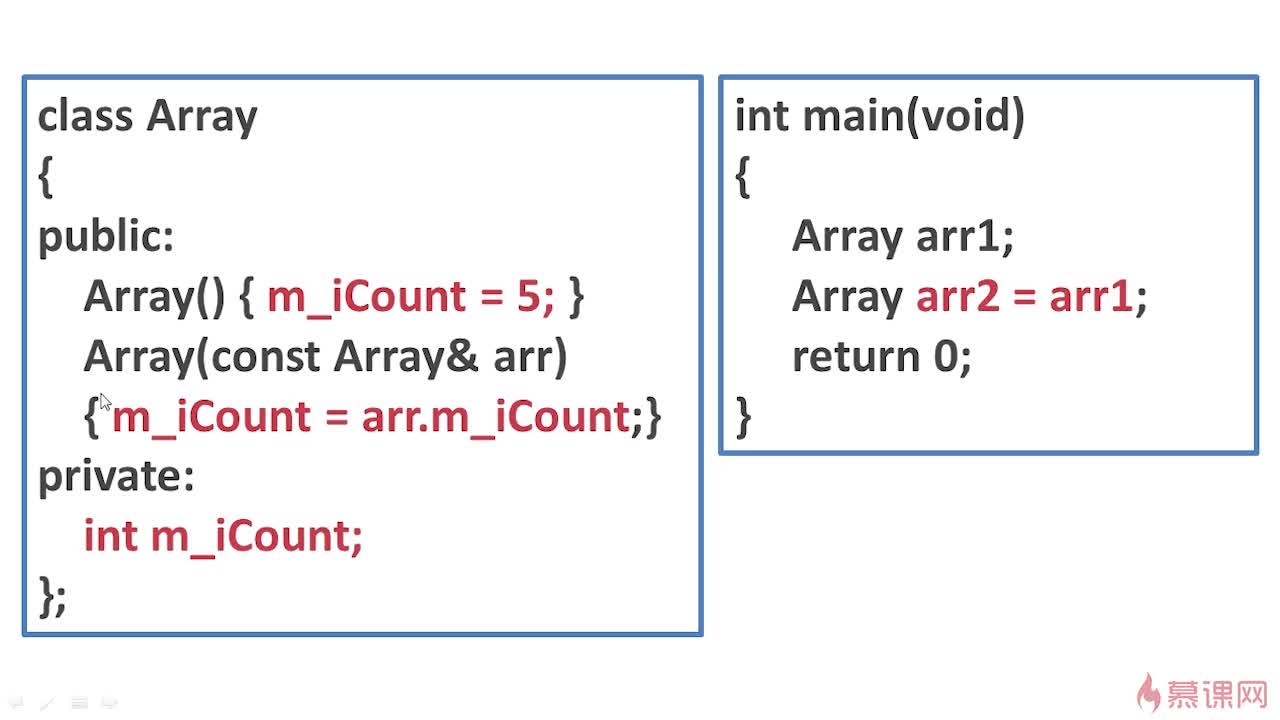
# 深拷贝与浅拷贝：

**深拷贝：将堆中内存的数据也进行拷贝（两个指针数据占用两块内存）**

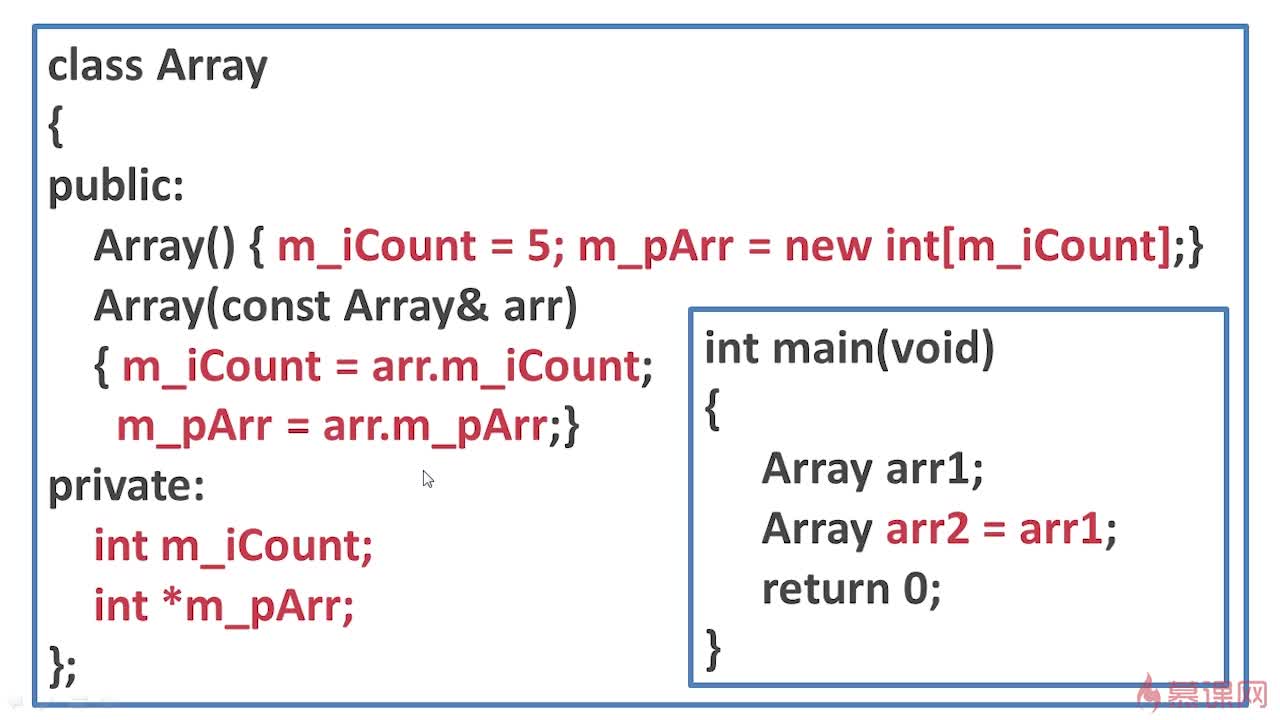
**浅拷贝：简单的数值进行拷贝**

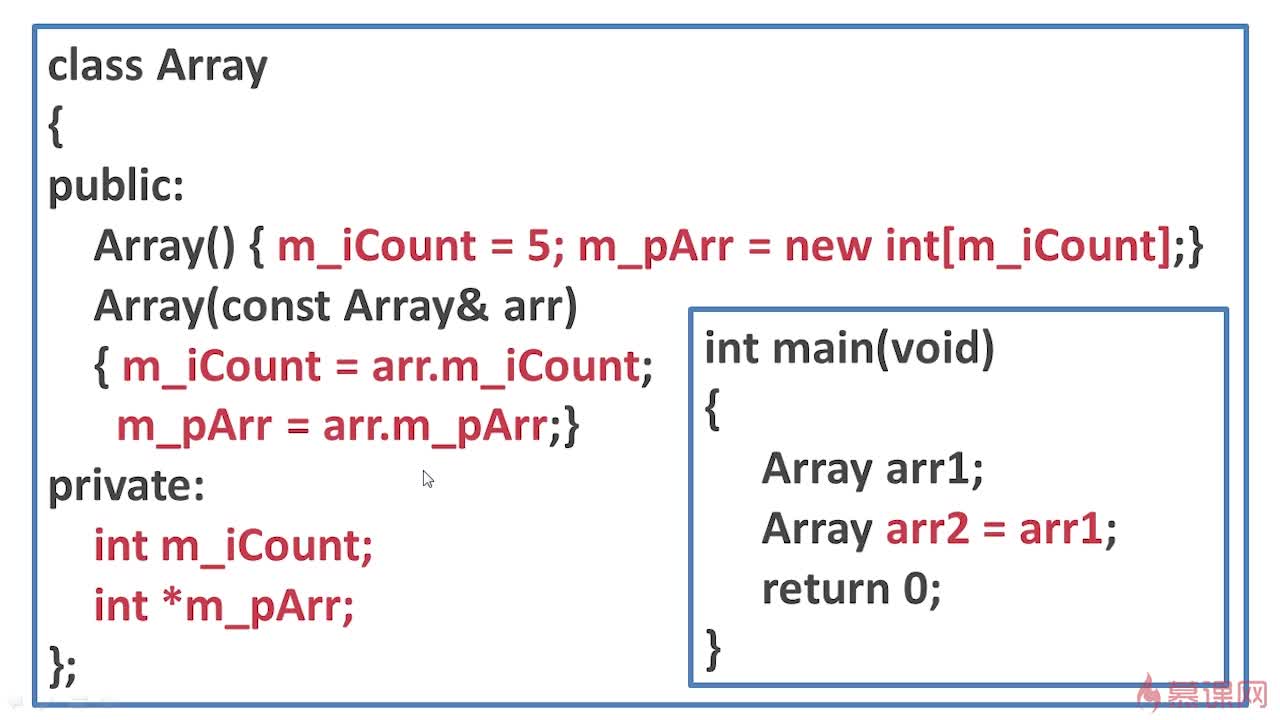
**例：（本例中Array是一个数组的类，参数表示数组的成员个数）**

**浅拷贝：**



**对于上面这种浅拷贝，它只是简单对进行了数值的拷贝，但是，对于下面这种使用浅拷贝的方式就会出现严重的问题：**





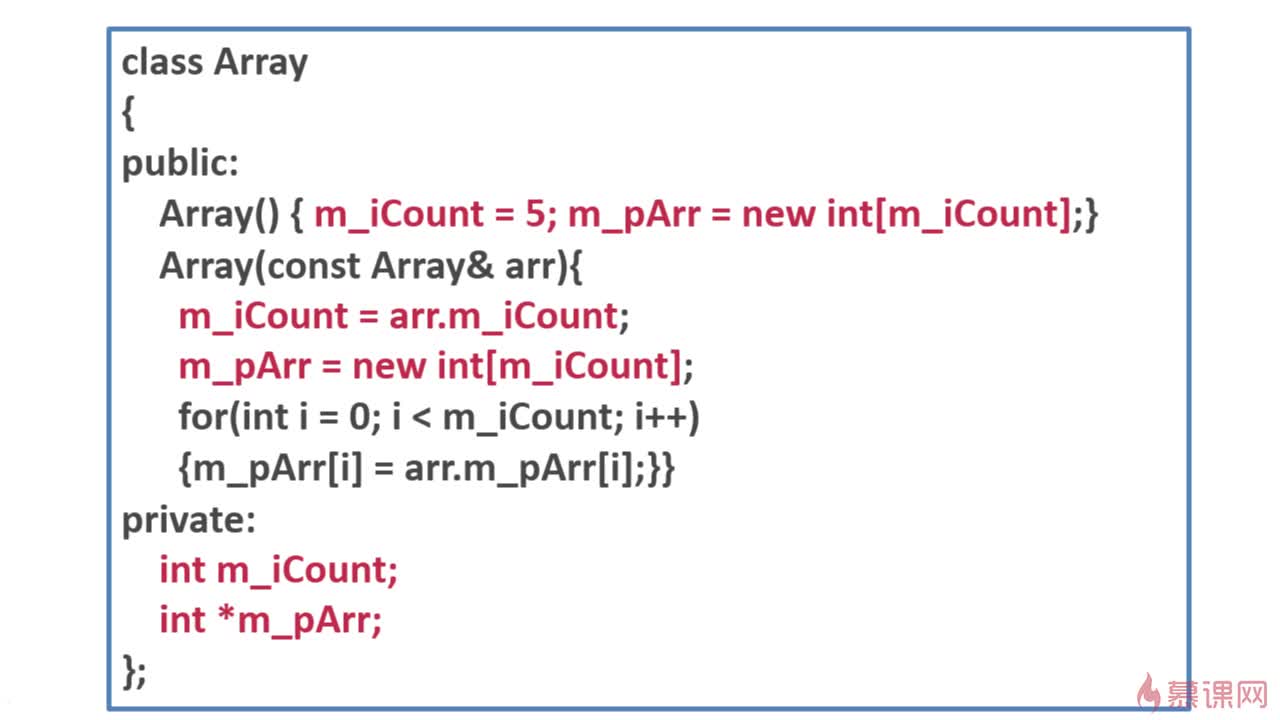
**存在问题：**

**1.在调用拷贝构造函数的时候两者指向同一块内存。**

**在arr2调用这块内存的时候，这块内存就会被重写而覆盖掉之前的arr1中的内存**

**2.在最终释放内存的时候由于arr1的这块内存已经为了防止泄露而被释放掉了一次，arr2的指向同一块内存，后来又被释放一次。同一块内存被释放掉了两次，此时计算机会崩溃。（程序崩溃：错误信息难以理解）**

**深拷贝：**



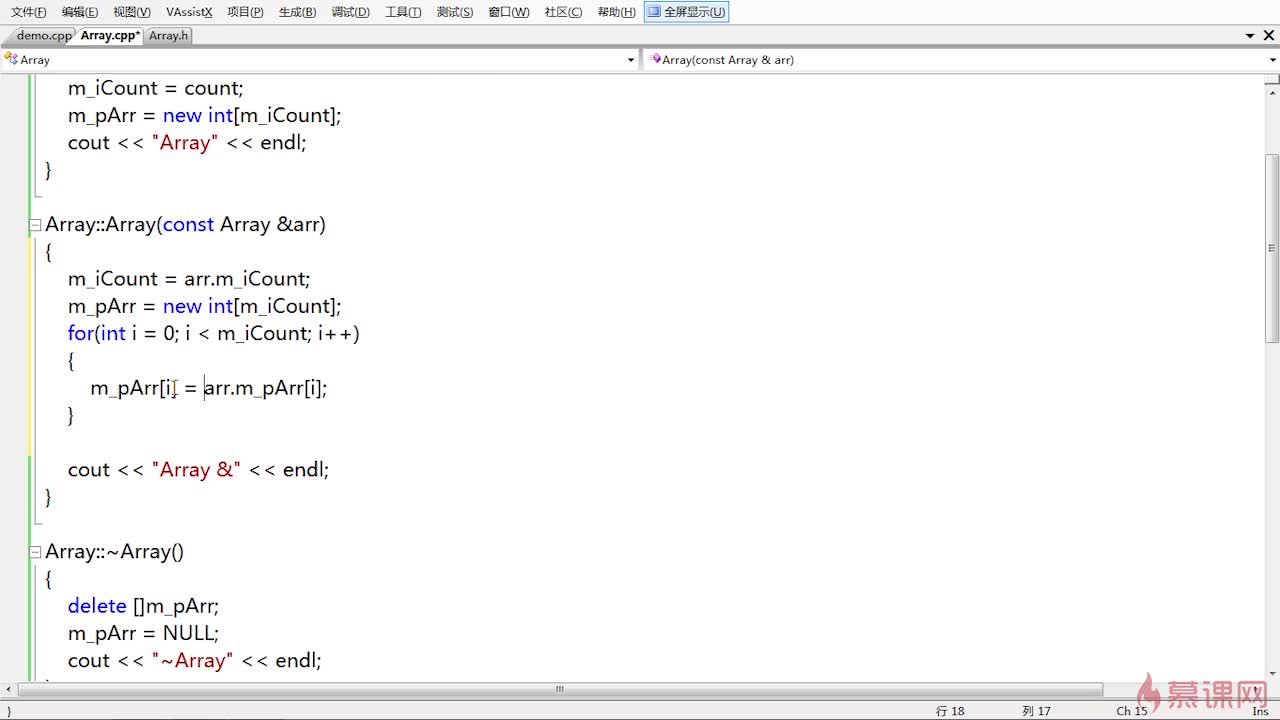
**这种拷贝方式将堆中内存的数据也进行了拷贝。**

**在这里再明确一个概念，在新定义一个变量时调用的是他的类的默认构造函数，在新定义了一个变量并用另一个变量对他进行初始化的时候调用的是拷贝构造函数**

**使用深拷贝的情况基本上是类中包含指针成员的时候**

**深拷贝的思路：在构造函数申请内存，在构造函数下面为内存赋值。**

**深拷贝：**



**当数据成员中含有指针时，浅拷贝会使两个对象的成员指针指向相同的内存地址**

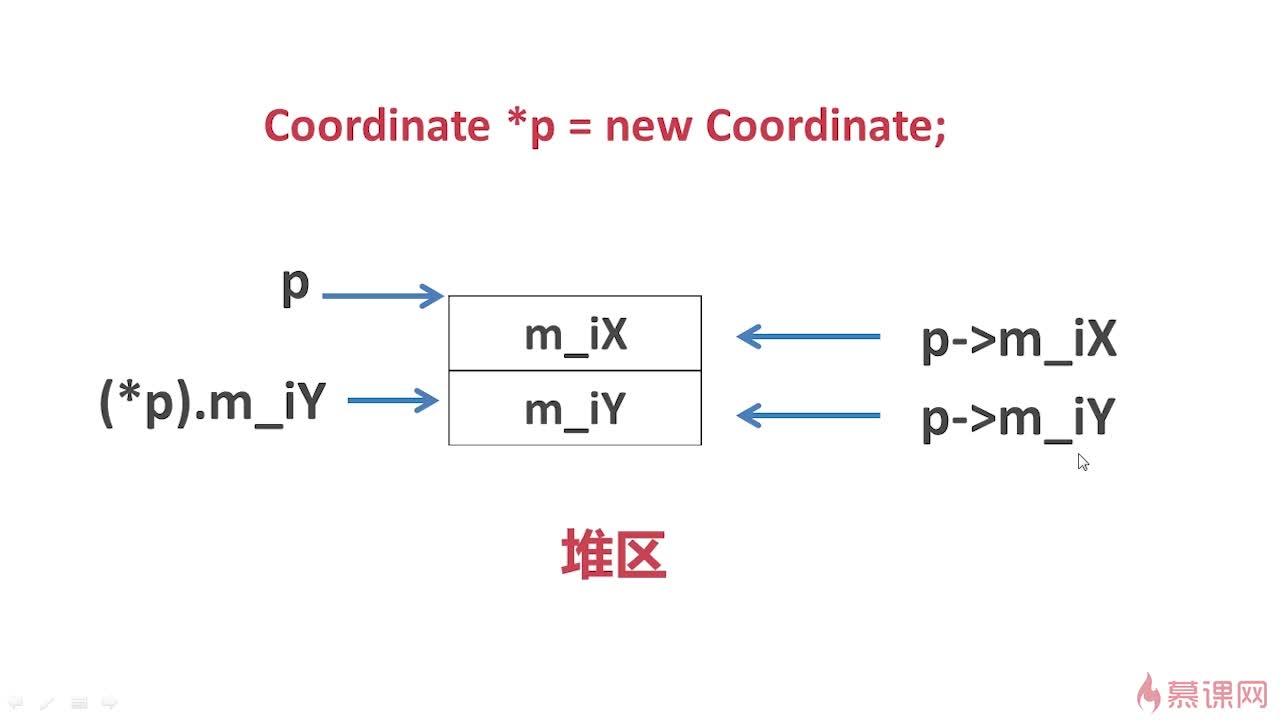
**深拷贝不是简单的值拷贝，而是将指针成员指向的内存数据也进行拷贝。**

**不是所有的对象都需要深拷贝，应该根据情况合理使用。**

# 对象指针

**定义：指向对象的指针成为对象指针**

**例:**



**访问方法：**

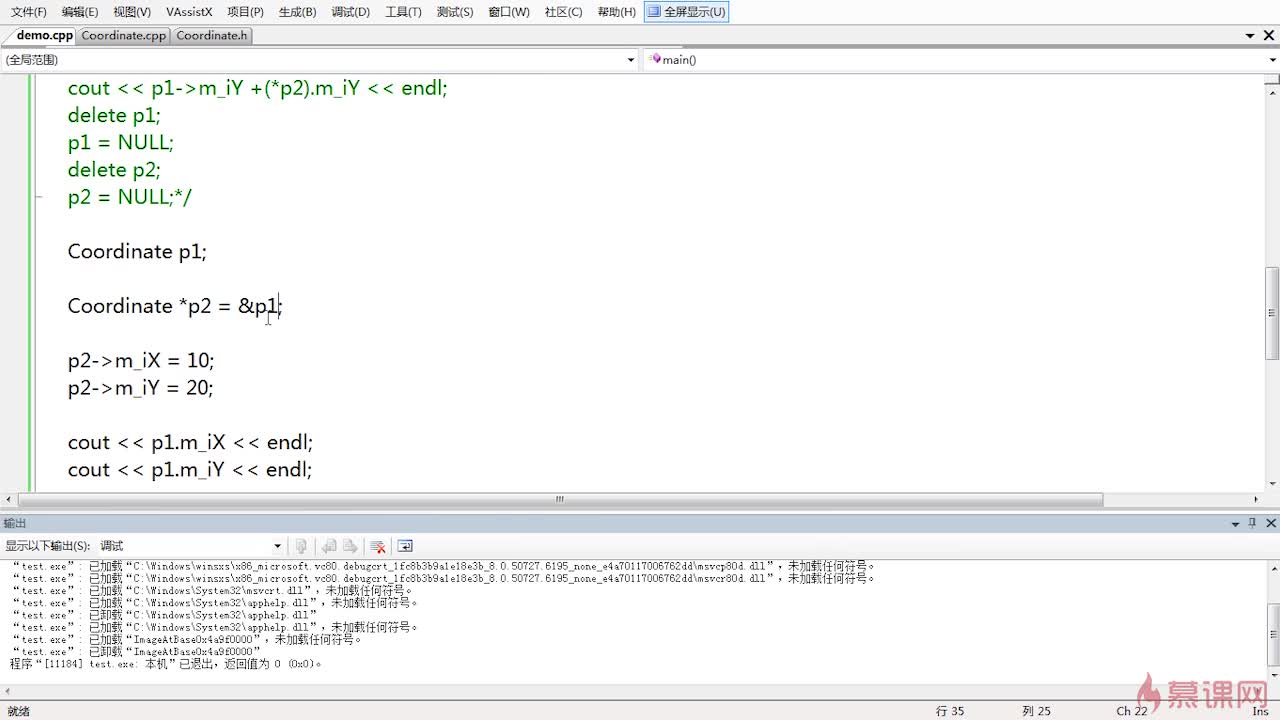
**p->m\_iX**

**或者(\*p).m\_iX**

**在栈中定义对象指针**

**Coordinate \*p2 = &p1;**

**使得p2可以指向p1的内存了**



**同样的访问限定符（如public）可以出现多次。用指针新建对象的时候，如果是默认的构造函数，后面的括号可以写可不写：**

**class Corrdinate**

**{**

**public:**

**Corrdinate();**

**~Corrdinate();**

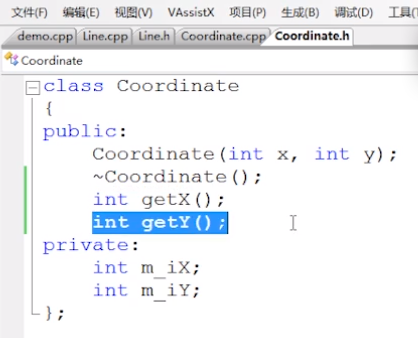
**public:**

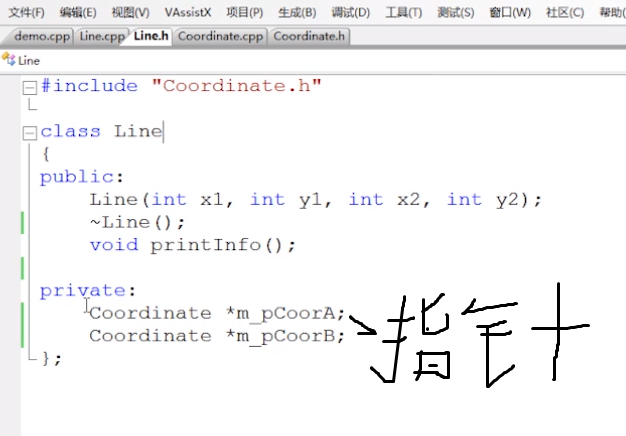
**int m\_iX;**

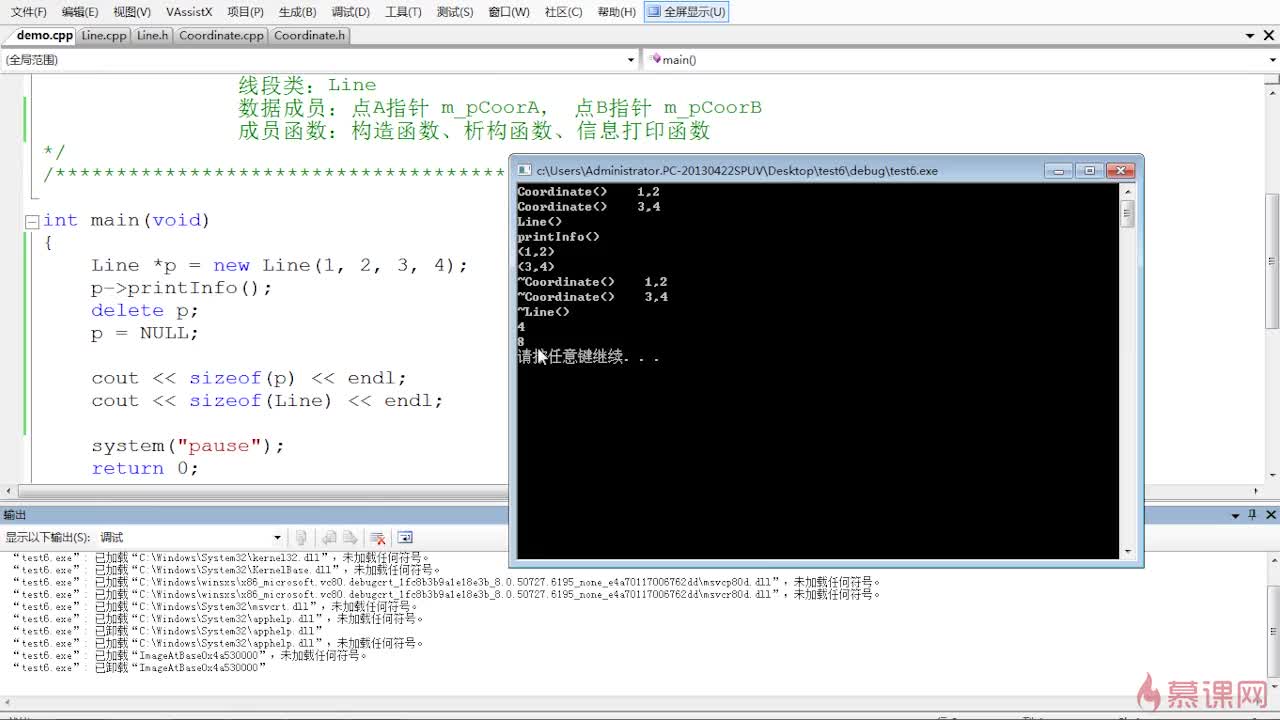
**int m\_iY;**

**};**

**例：**







**sizeof（Line）占8个内存单元**

**sizeof (p)占4个内存单元**

**原因：（1）32位的编译环境，一个指针占4个字节，1一个字节就是一个基本的内存单元，1Byte = 8 bit,即一个内存单元里可以存储2的8次方不同情况的高低电平。**

**（2）因为Line这个类中有两个对象成员指针，所以占了8个字节。**

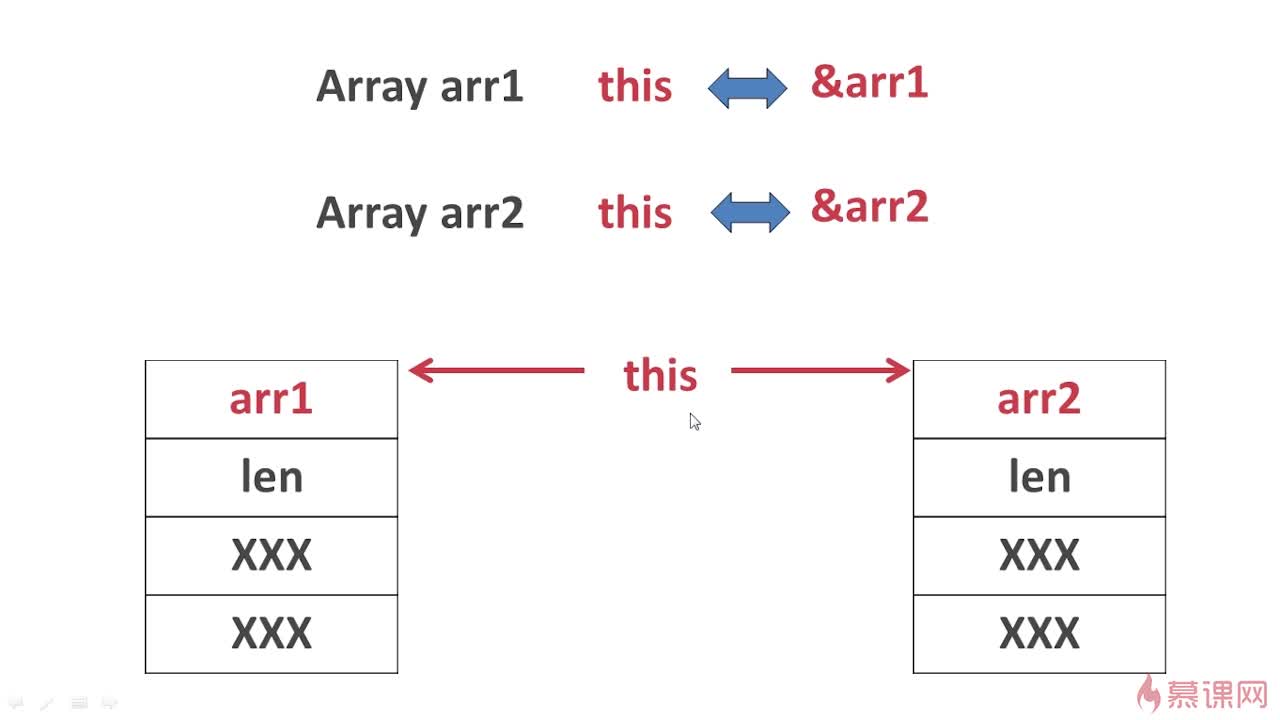
# This 指针

**this指针存的即当前对象的地址，可以访问到自身对象的数据成员**

**This 代表当前自身的对象，谁调用 Array 构造对象，然后 This 就代表那个对象取代那个对象，也就是 Array arr1；**

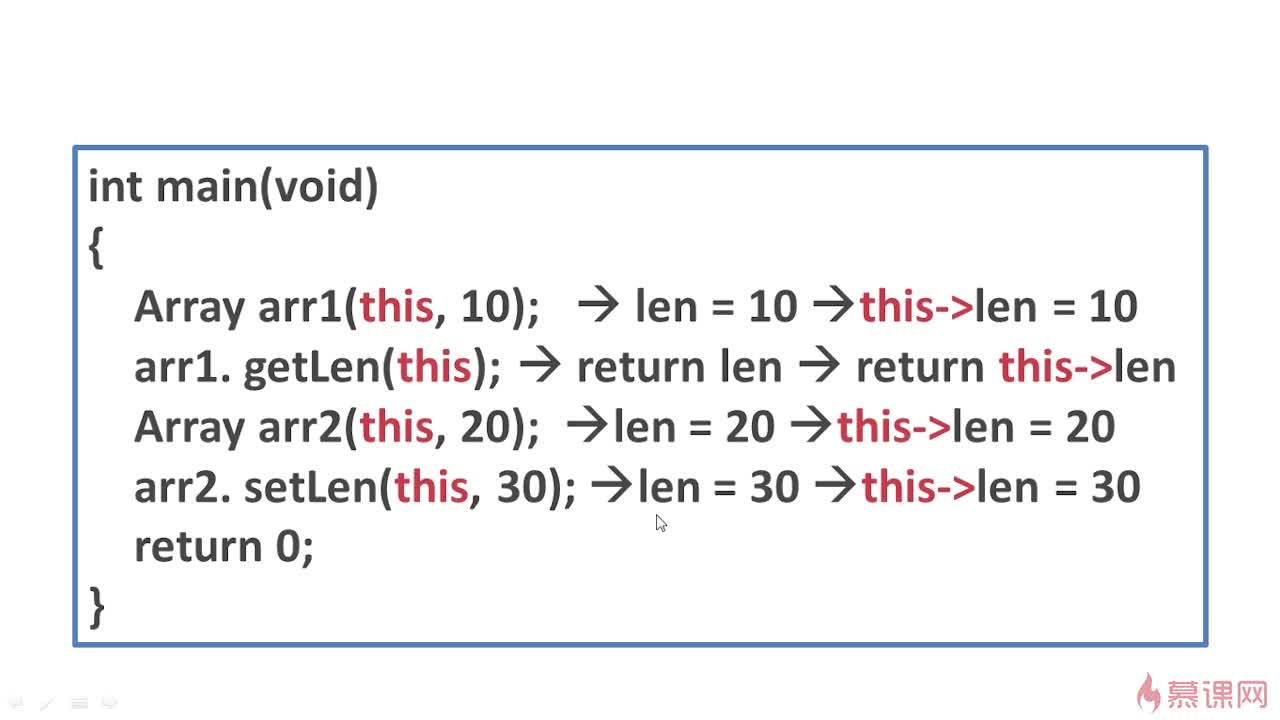
**对象各自的this指针指向各自对象的首地址，所以不同对象的this指针一定指向不同的内存地址**

**Array arr1；arr1.setLen（5）的时候，{this -> len = len} 中 this 就是代表了 arr1 对象，会取代 this ，而 this 其实是对象的地址，也就是指针，{this -> len = len} 就代表是 { arr1.len = len; },从而标记区别了数据成员和参数。**

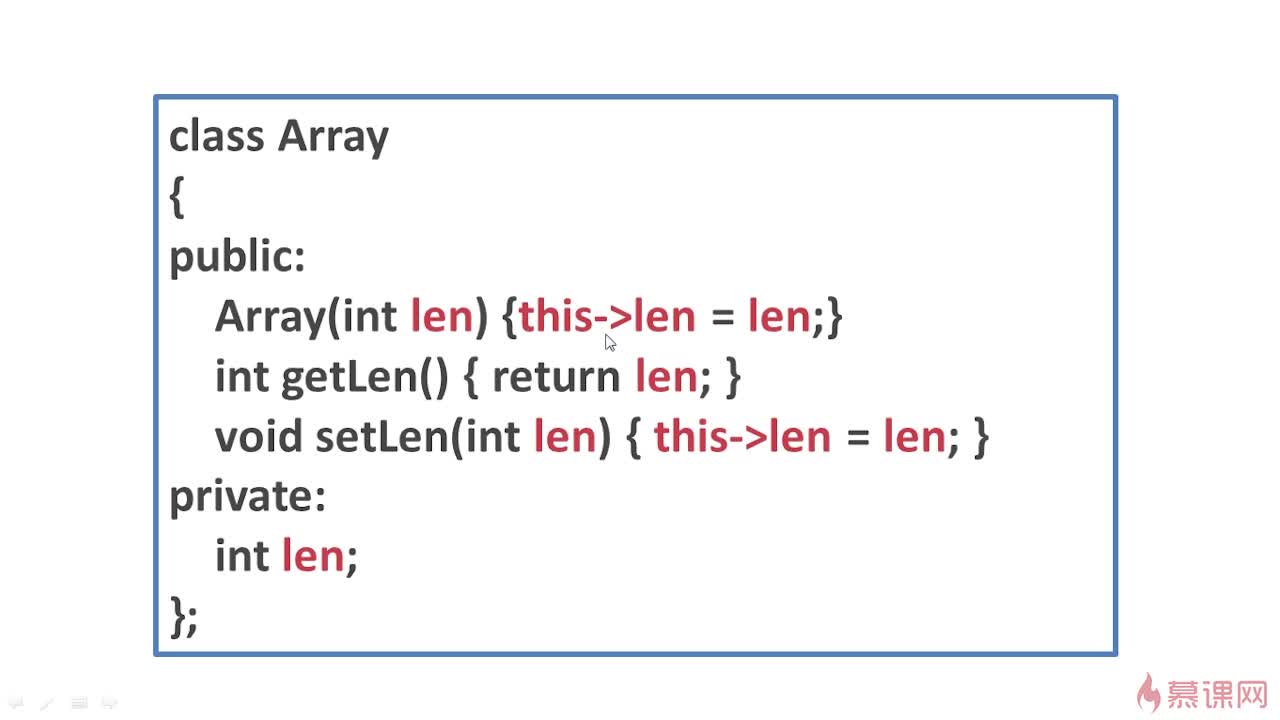


**每次调用成员函数时其实都是用到了this 指针，**

**所以编译器会自动帮我们完成这一步（因为所有的都会用到 this）**



**同时，若形参和实参的名字一样，则在赋值的时候系统就不知道到底是把形参的值赋给实参还是把实参的值赋给形参，这时就需要用到this指针：**



**注意上面this指针的写法：this-〉len.**

**Array Q()//这是一个函数名，返回值为Array型的**

**要求返回值类型是一个对象，而**

**Array Array::Q()**

**{return \*this;}**

**this为指针，\*this 即谓对象。**

**如此返回的对象\*this是一个临时的对象，并不是主函数中实例化的arr1，对该返回的Array进行修改并不能影响原来的arr1.**

**如果想在arr1上进行修改需要加入引用符号即**

**Array& Array::Q()**

**{return \*this;}**

**如此可在调用函数过后急需加.进行调用其它成员函数。**

**也可直接返回指针**

**\*Array Array::Q()**

**{return this;}**

**在调用函数后加->同理同时调用对象中其他函数。**

**在C++中，this指针被隐含地声明为: X \*const this,这意味着不能给this 指针赋值；**

**在X类的const成员函数中，this指针的类型为：const X\* const, 这说明this指针所指向的这种对象是不可修改的（即不能对这种对象的数据成员进行赋值操作）;**

**例：**

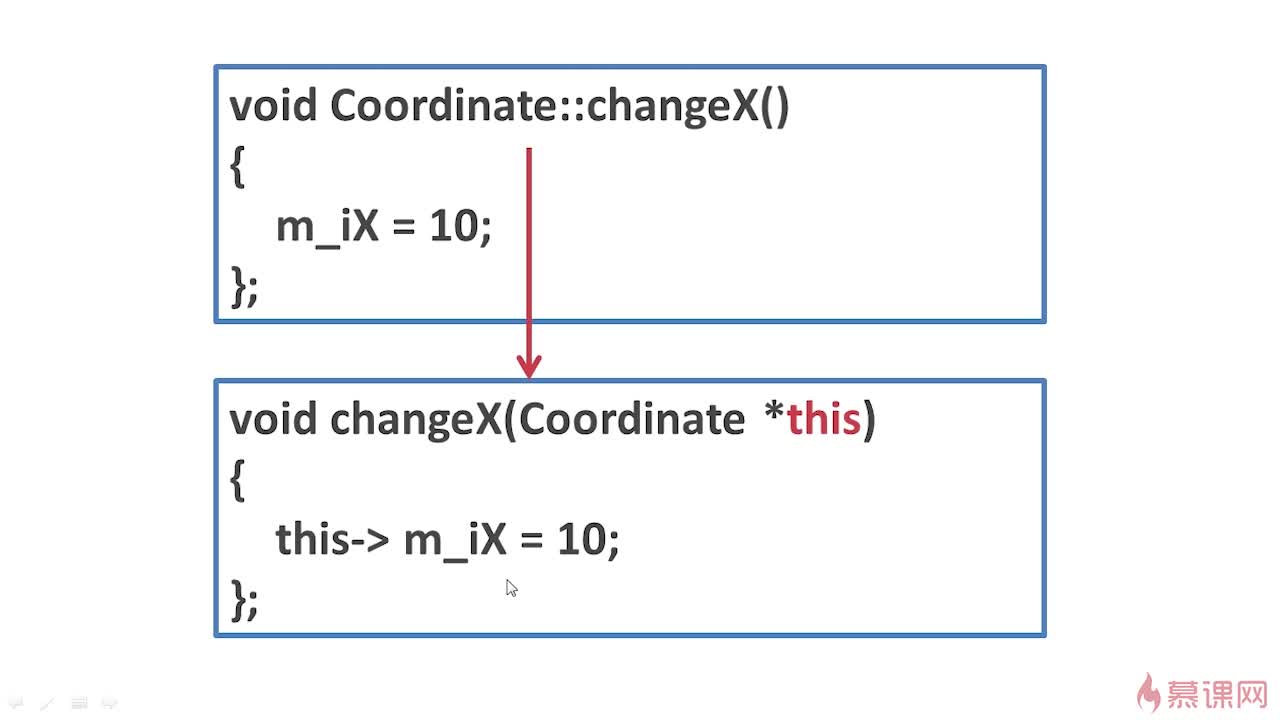
**因为对于c++的成员函数（当然不是静态成员函数),都会含有一个隐藏的参数，对于上例A中的int GetValue()函数，在编译后会变成：**

**int GetValue(A \* const this);  //不能修改this变量，但可以修改this指向的内容，即：this是常量指针。**

**而对于int GetValue()const ，编译后是：**

**int GetValue(const A\* const this);**

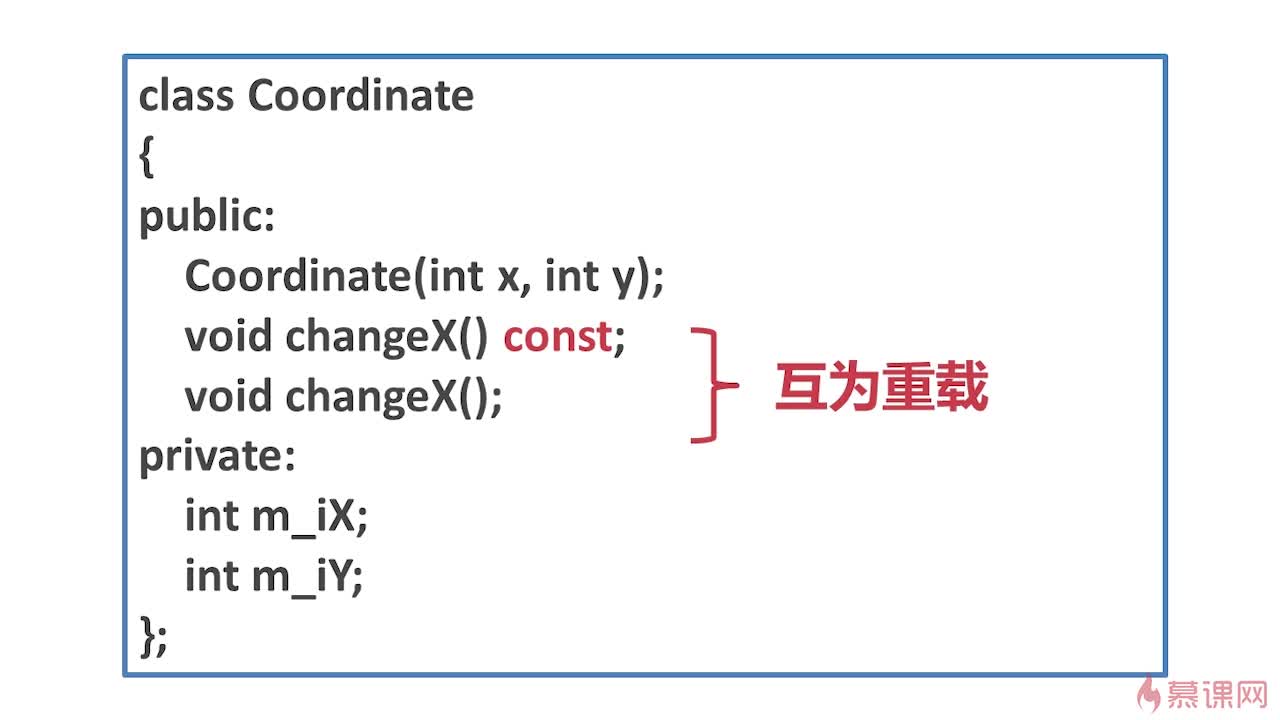
**5：对于普通（非常成员函数）的成员函数， 内容隐含了当前对象this的参数**



## const重现江湖

### 常对象成员和常成员函数:

**互为重载：**

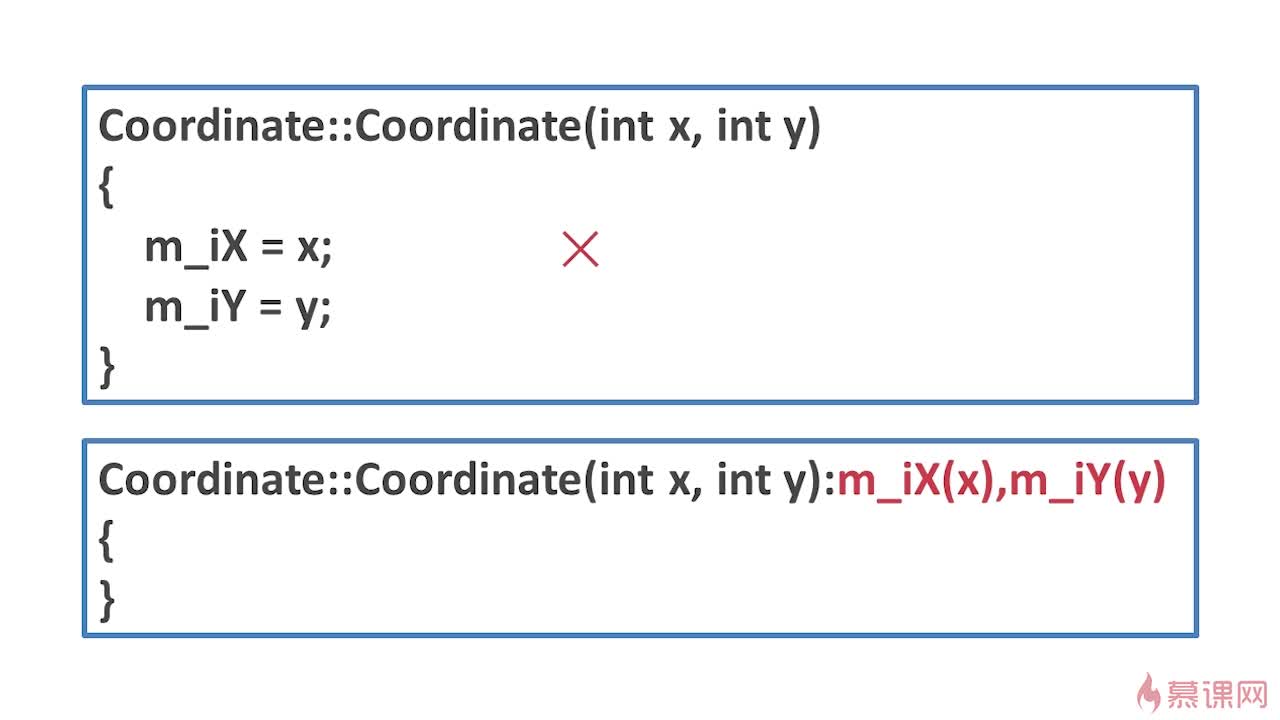


**当数据成员为const修饰的时候，比如**

**const coor m\_iX;**

**const coor m\_iY;**

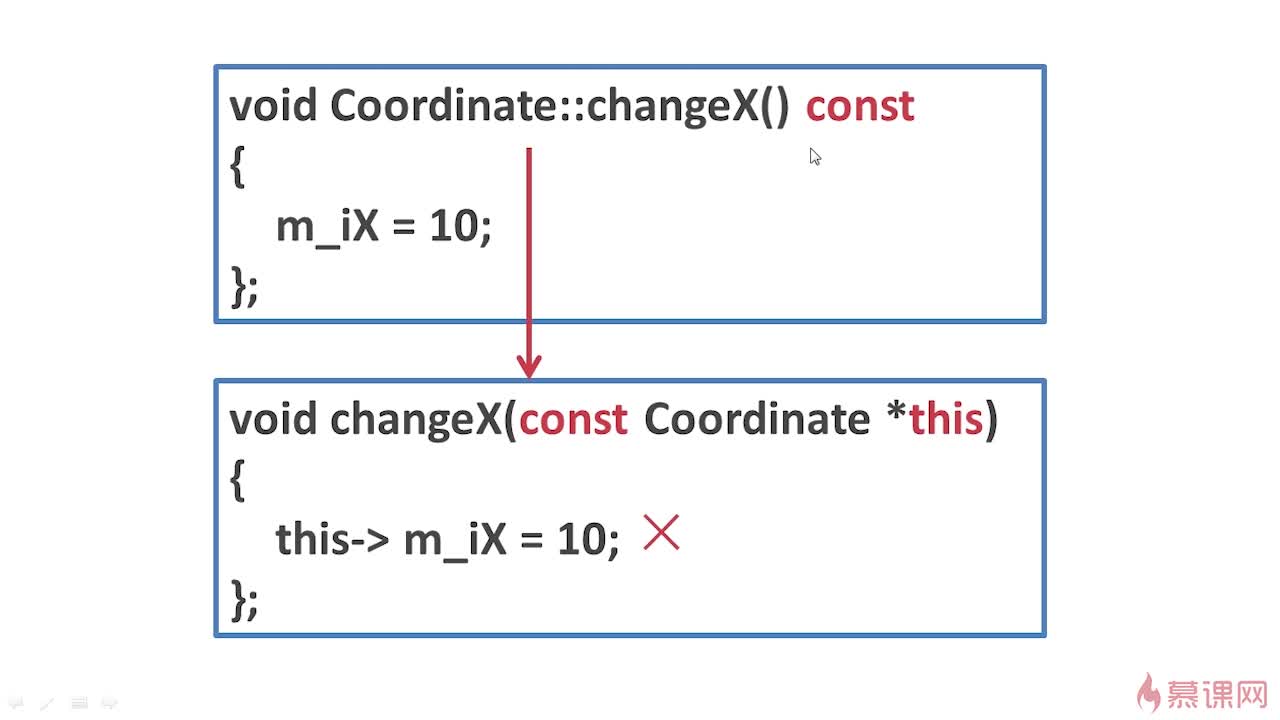
**的时候，要有初始化列表初始化：**



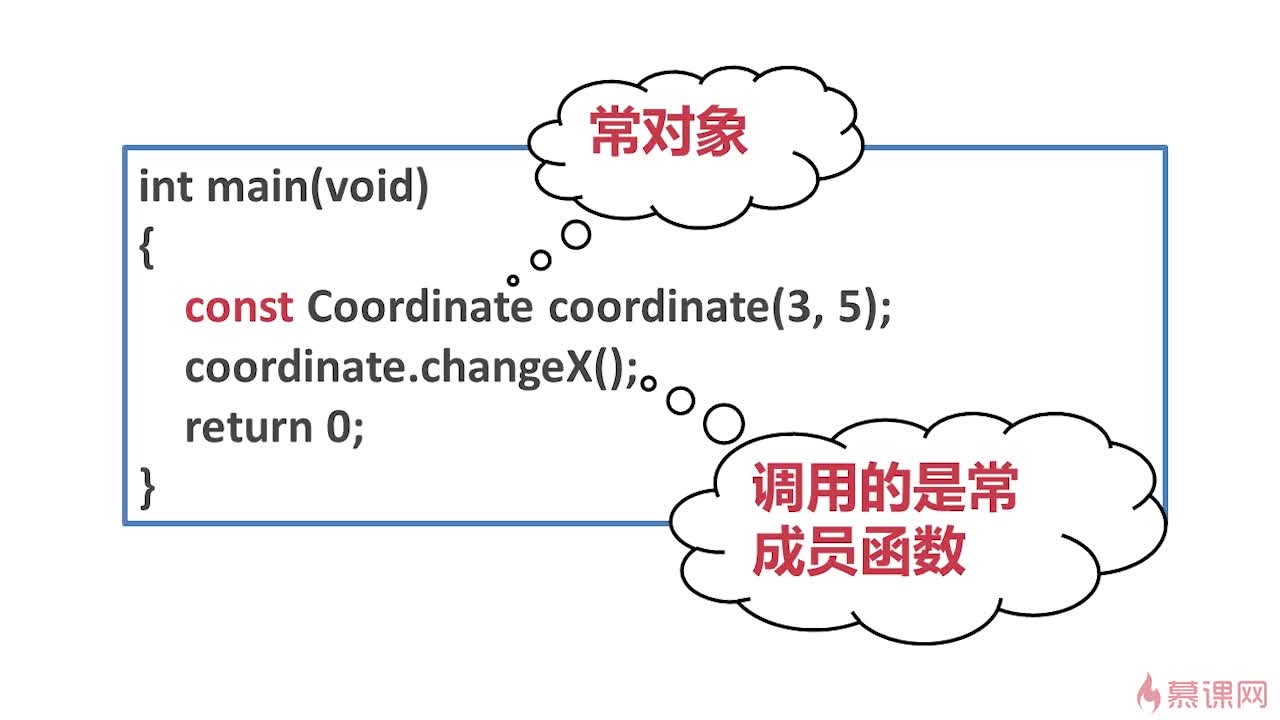
**常成员函数不可以改变数据成员的值：**

**因为此时传入的参数（对象指针this）是一个常指针，不能通过此常指针修改数据成员的值**

**const Coordinate \*this**



**想要调用const修饰的常成员函数，就必须用const定义一个常对象，常对象调用常成员函数**



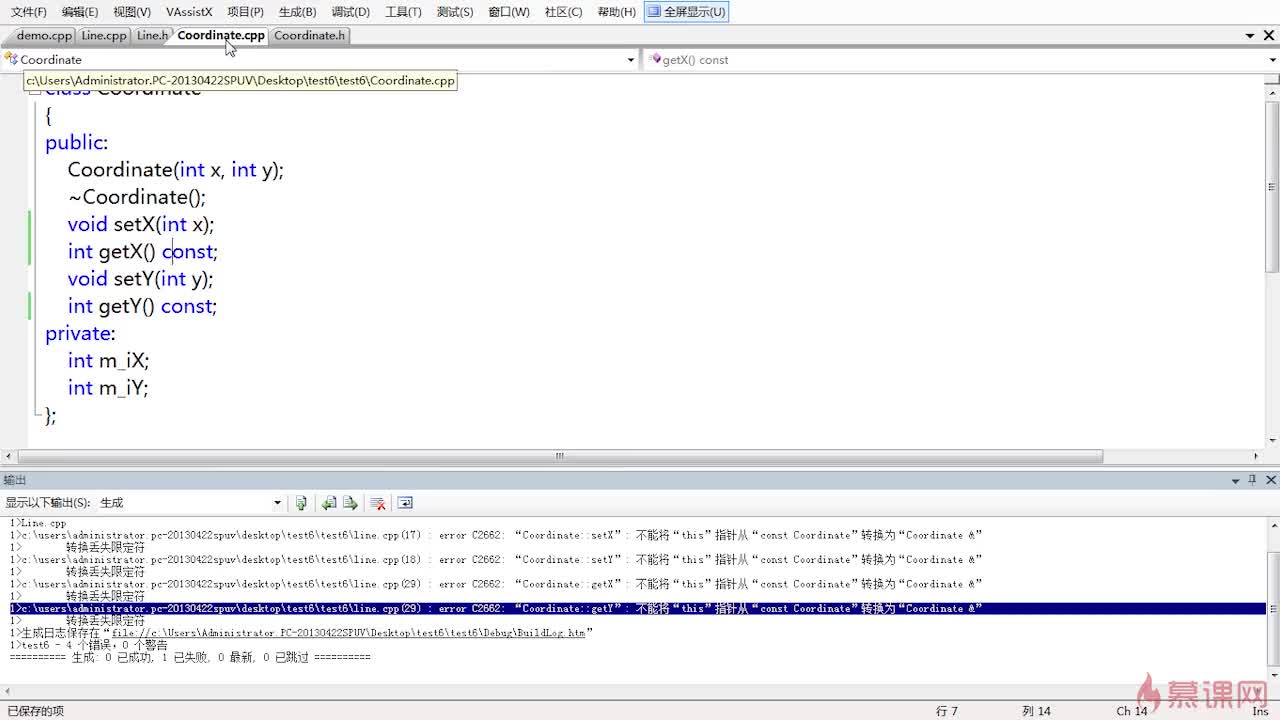
**当一个成员函数被调用时，自动向它传递一个隐含的参数，该参数是一个指向这个成员函数所在的对象的this指针**

**如果一个对象中有的两个相同名字成员函数，其中一个是普通成员函数，另一个是常成员函数，那么在调用时的规则如下：**

**定义的普通对象会默认调用普通成员函数，但是，普通对象也可以调用常成员函数，定义的常对象只能调用常成员函数**

**原因：常成员函数要求传入的对象的this指针是具有“只可读”权限的，而普通的成员函数要求传入对象的this指针是有“既可读又可写”权限的，所以，当常对象调用普通函数时，相当于将只具有读权限的this指针传给了要求传入既可读又可写的this指针的函数，这样肯定是不行的，但是反之，将普通对象的this指针传给常成员函数却是可以的，这就体现了所谓的“权限大的可以给要求权限小的，但是权限小的不能给要求权限大的”**

### 常成员函数实例



**要注意，对于setX这种改变对象成员的函数，是不能定义为常成员函数的，因为如果定义成常成员函数，他就不能改变对应的对象成员的值，但是像getX这类不改变对象成员的函数，我们就可以将它们设定为常成员函数。**

**如何把一个普通的成员函数改成一个常成员函数？**

### 常成员函数修改

**声明部分：在后面写上const，**

**int getX() const;**

**定义部分: 末尾也要加上const**

**int Coordinate::getX()const{..}**

**const和前面的函数签名要有空格**

**常成员函数的本质是内部使用this指针**

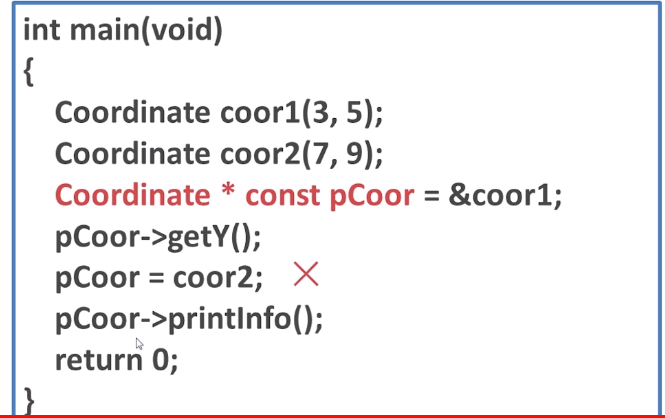
**常成员函数内使用数据成员时，不能改变数据成员的值。**

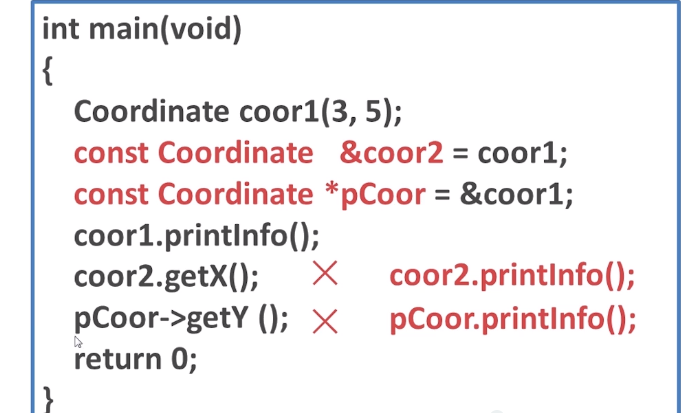
**常成员函数中可以使用普通的数据成员，但是不能改变对象成员的值**

**常成员函数中不能调用普通成员函数**

**对象引用：Coordinate &coor2=coor1;**

**对象常引用：const Coordnate &coor2=coor1;**





**注意：**

**1.在使用对象引用的时候，可以将这个对象直接赋值给对象引用**

**2.在使用对象指针的时候，必须对对象进行取地址的操作**

**3：注意以上两种用法的差别，第一种定义的const修饰的是pCoor,意思是该指的指向不可以改变，所以\*pCoor=&coor1以后再使用pCoor=coor2就会出错，编译器就会报错。而第二种const修饰的是\*pCoor,意思是该指针的对象的值可以改变，但是，该指针指向的对象不可以改变，所以，\*pCoor=&coor1以后再使用pCoor->getY就会出现错误。**

**总结：**

**（1）：一个对象可以有多个对象引用**

**（2）：常对象只能调用常成员函数，不能调用普通成员函数**

**（3）：普通对象能够调用常成员函数，也能够调用普通成员函数**

**（4）：常指针和常引用都只能调用对象的常成员函数。**

**（5）：使用常指针和常引用后都不能改变对象的值，因为一旦使用了const，对应的指针和引用的this指针都会变为“只读”权限的，而想要改变值。就必须要有“可写”权限。**