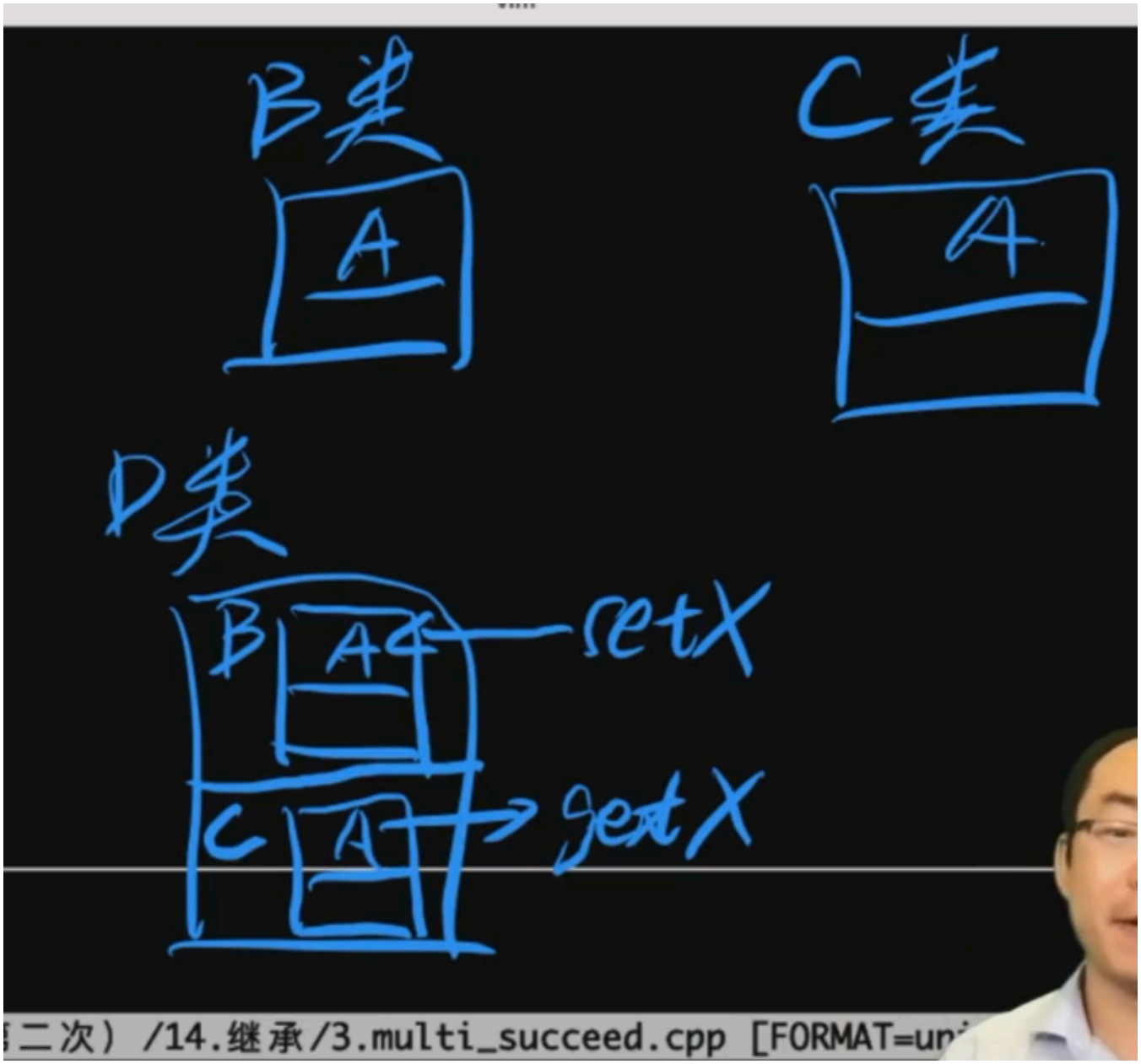
多继承可能产生的问题：

```
#include<iostream>
using namespace std;

#define BEGINS(x) namespace x {
#define ENDS(x) }

BEGINS(test1)
class A {
protected:
    A() : x(9973) {}
    int x;
};
class B : virtual public A {
public:
    void setX(int x) {
        cout << "Set x : " << &(this->x) << endl;
        this->x = x;
        return ;
    }
};
class C : virtual public A {
public:
    int getX() {
        cout << "Get x : " << &(this->x) << endl;
        return this->x;
    }
};
class D : public B, public C {};
int main() {
    D d;
    cout << d.getX() << endl;
    d.setX(10000);
    cout << d.getX() << endl;
    return 0;
}

ENDS(test1)


int main() {
    test1::main();
    return 0;
}
```

子类继承了父类所有的东西，前半部分是父类，后半部分是子类，画出如下的示意图：



看D类的图就知道，我们setX（10000）设置的是上面那部分内存，但是getX()获取的却是下面那片内存的值。他们设置的根本就是两个地址的变量，当然会出问题了。

底层知识不是一本C++语法书能够解决的。

如何解决？——虚继承

在B和C继承自A类的时候是虚继承：在继承的前面加上关键字**virtual**，相当于告诉编译器，一旦你发现有相同的A类部份了。麻烦帮我进行合并。

通常情况下不建议多继承，在真正程序开发过程中，没必要用这种所谓的多继承。而是提倡：**单继承实体类，多继承接口类（抽象类）。**结论：像上述发生的情况在工程中是一定可以避免的。

实体类：能产生对象的类。
接口类：不能单独产生对象。（不能产生对象不等于是接口类，因为像下面这种情况也是不能产生对象，但不是接口类）

```
//设计一个不能产生对象的类
class NoObject {
public:
    NoObject() = delete;
    NoObject(const NoObject &) = delete;
};


int main() {
    NoObject *p = (NoObject *)malloc(sizeof(NoObject));
    // NoObject b(*p);
    return 0;
}
```

那继承到底有什么用呢？——介绍一种功能类。

功能类：标记每一种类的性质。配合上重载就可以针对性地处理每种性质的类。

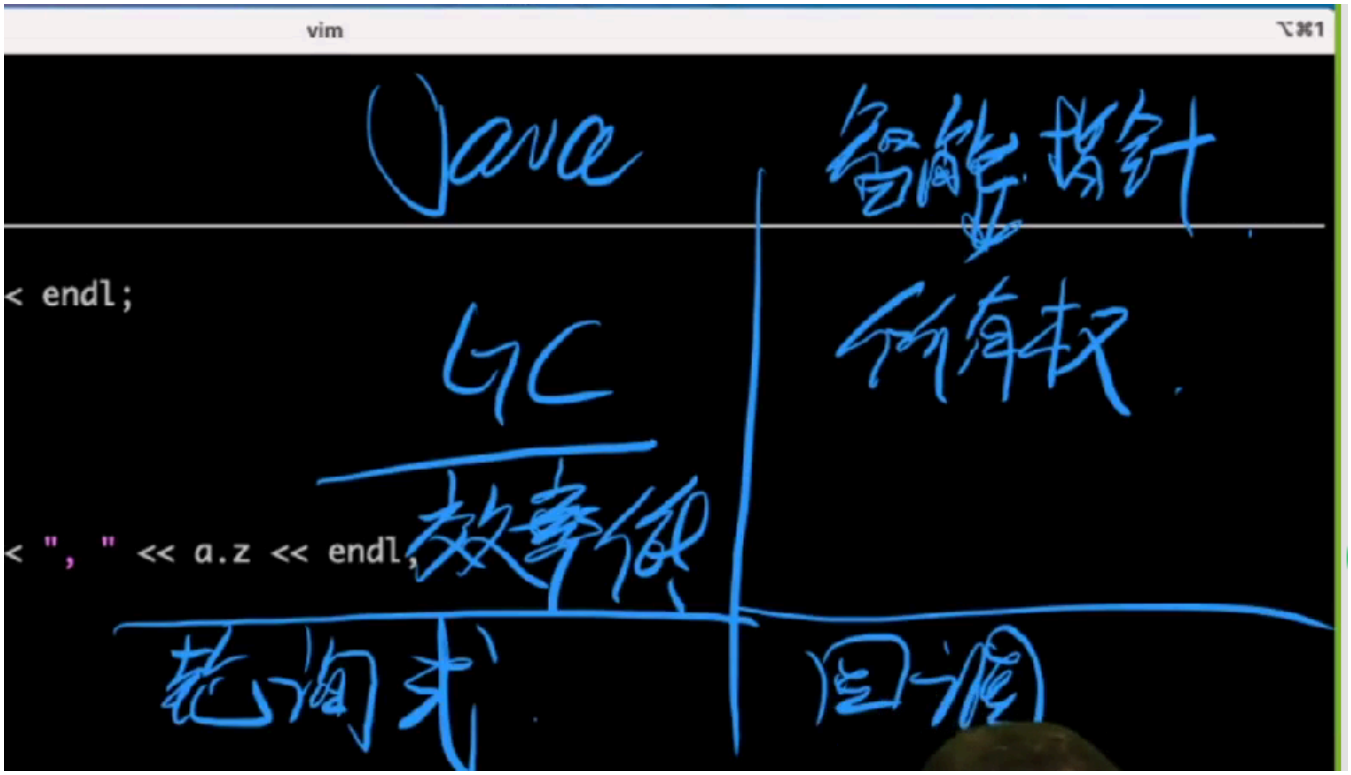
```cpp
//设计一个不能被拷贝的功能类；（拷贝构造+赋值运算）
class UNCOPYABLE {
public:
    UNCOPYABLE(const UNCOPYABLE &) = delete;
    UNCOPYABLE &operator=(const UNCOPYABLE &) = delete;
    UNCOPYABLE &operator=(const UNCOPYABLE &) const = delete;
protected:
    UNCOPYABLE() = default;
};
class A : public UNCOPYABLE {};

int main() {
    A a;
    A b;
    // b = a;  // no, operator= delete
    // A b(a); // no, copy constructor delete
    return 0;
}
```

萃取类的技术：

```
class HAS_XY {
public:
    int x, y;
};

class HAS_XYZ : public HAS_XY {
public:
    int z;
};

class A : public HAS_XY {
public:
    A() { x = y = 1; }
};
class B : public HAS_XY {
public:
    B() { x = y = 2; }
};
class C : public HAS_XYZ {
public:
    C() { x = y = z = 3; }
};
class D : public HAS_XY {
public:
    D() { x = y = 4; }
};
class E : public HAS_XYZ {
public:
    E() { x = y = z = 5; }
};

void func(HAS_XY &a) {
    cout << "has xy : ";
    cout << a.x << ", " << a.y << endl;
    return ;
}

void func(HAS_XYZ &a) {
    cout << "has xyz";
    cout << a.x << ", " << a.y << ", " << a.z << endl;
    return ;
}
int main() {
    A a;
    B b;
    C c;
```

```
    D d;
    E e;
    func(a), func(b), func(c), func(d), func(e);
    return 0;
}
```

垃圾回收策略



基于回调：被动执行。（智能指针，count=1时，自动就回收了），效率高。
轮询式：主动等待，效率慢

不要当简单的知识点学习，学习方法就是：从实现原理方面理解知识。所以记笔记真正应该记什么？——即如何面对一个知识点，如何对知识点进行更深入地学习和思考。这是这套学习方法中包含地，看STL源码只是其中一个环节，但是你自己还得去实现呐。