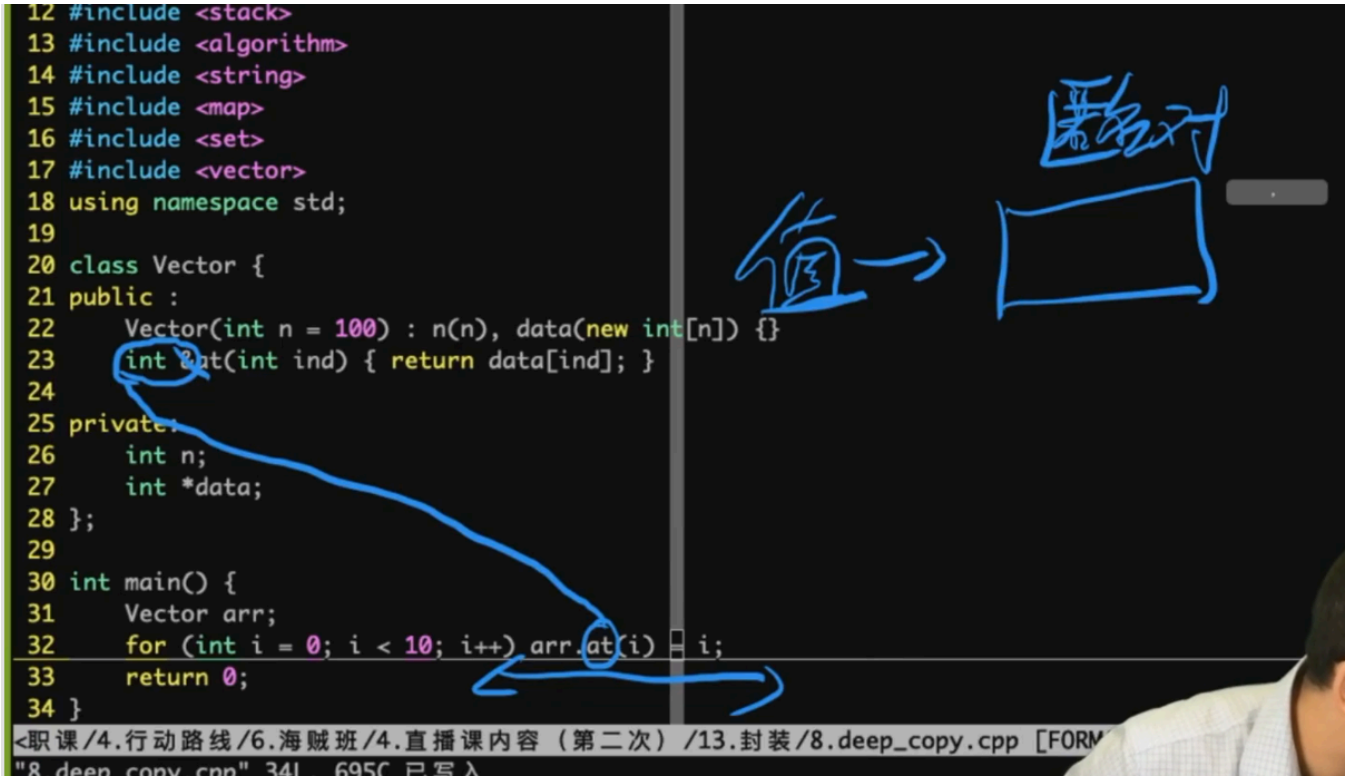
> 在什么情况下，会涉及到深拷贝，浅拷贝？什么时候需要定制化所谓的拷贝构造函数？

左值返回引用

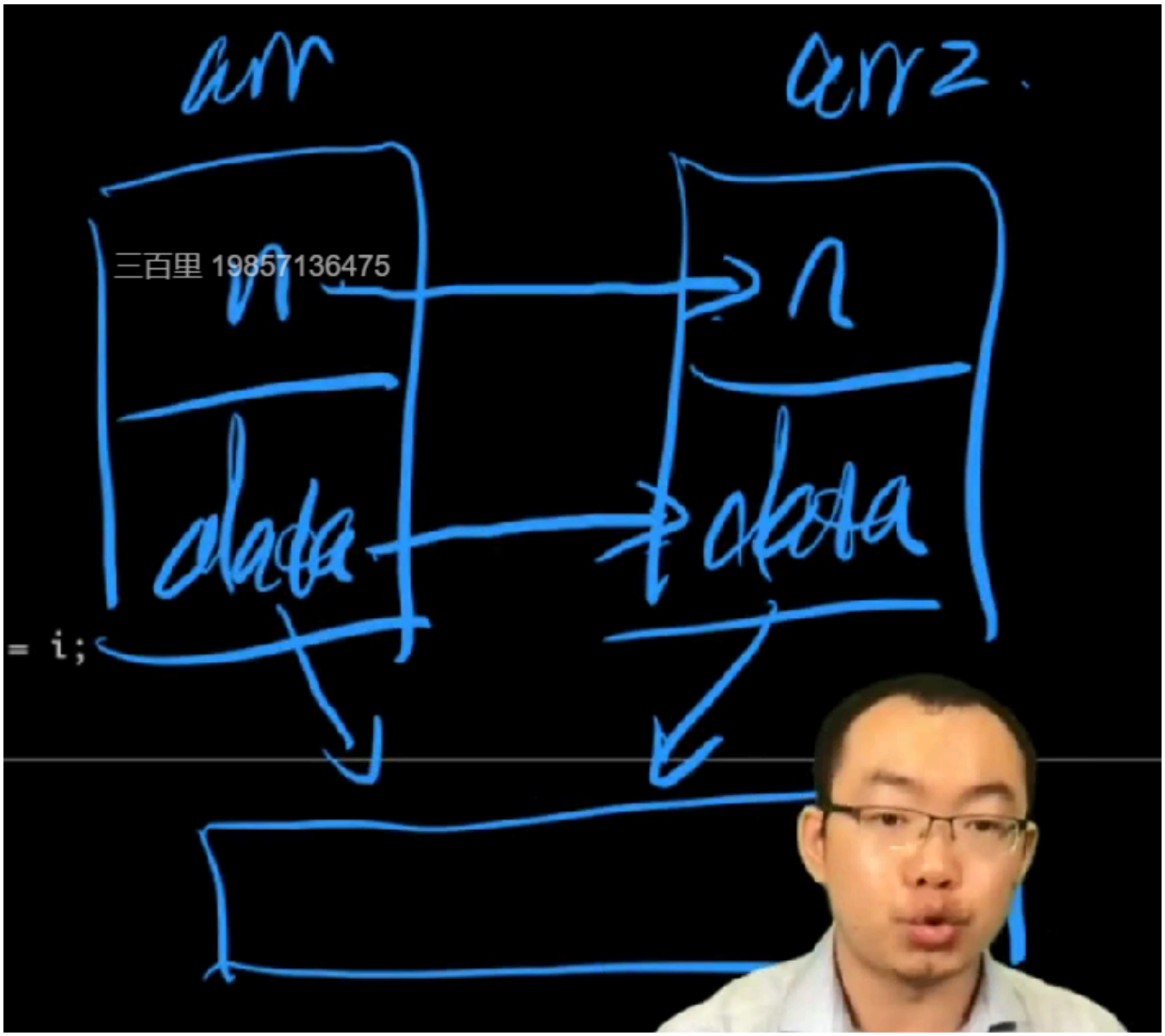


- 如果at()函数，返回的是一个数值，而不是引用，那么编译器会拷贝给一个临时的匿名对象，
- 在赋值语句里，变量i想要赋值给一个匿名对象，这个是不对的，会报错。

浅拷贝

按理来说arr和arr2应属于不同的两个对象，arr2的变化不应该影响到arr的变化，但是结果并非如此。这个就是浅拷贝。

原因：**默认**的拷贝行为，**data指向同一片数据区。**



- **浅拷贝**可以解决**大多数情况下的需求**。
- 存在**指针字段**的拷贝就需要**深拷贝**。

如何完美实现深拷贝？：8.deep_copy.cpp

有一种new函数叫做**原地构造**：给个地址，把对象构造进去

```
new(地址) 对象 // 中间放上地址：在当前地址上去调用某个构造函数
```

new需要被构造对象有默认构造函数。

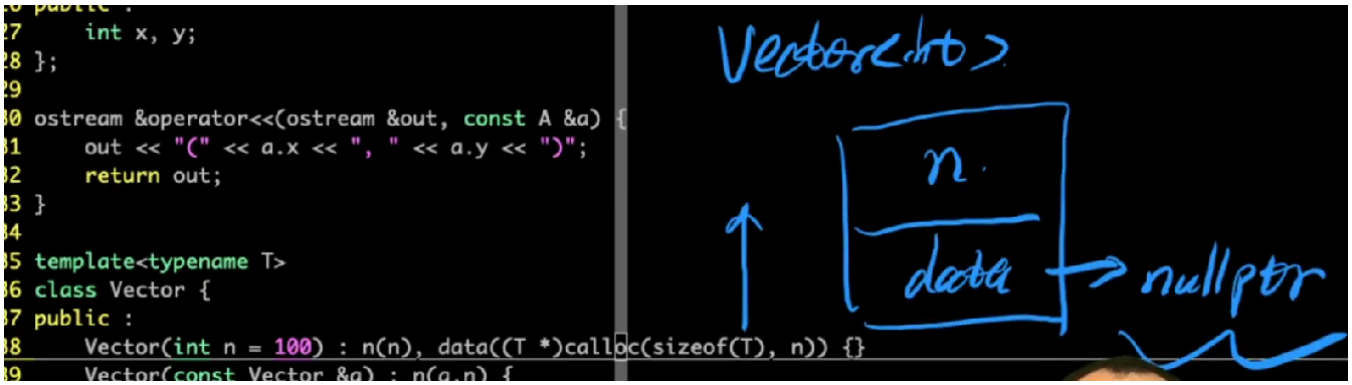
解决办法是：

```
(T *)calloc(sizeof(T), n);
//也就是说，我先把空间开辟出来，但我对里面的初始化先不轻举妄动。
```

```cpp
#include<iostream>
#include<cstring>
using namespace std;
#define BEGINS(x) namespace x {//测试样例
#define ENDS(x) }

BEGINS(haizei)
class A {
public:
    int x, y;
};

ostream &operator<<(ostream &out, const A &a) {
    out << "(" << a.x << ", " << a.y << ") ";
    return out;
}

template<typename T>//引进模板，并把data存储的类型改为任意的T
class Vector{
public:
    Vector(int n = 100) : n(n), data((T *)calloc(sizeof(T), n)) {}
    Vector(const Vector &a) : n(a.n) {
        /*

        // 一、普通做法：简单复制
        data = new T[n];
        memcpy(data, a.data, sizeof(T) * n);
        // 这样写究竟有什么问题？答：data里面可能不止存整型那么简单，如果data里的对象也需要深拷贝的时候,比

        // 二、改良做法：原地构造：
        data = new T[n];
        for (int i = 0; i < n; i++) {
            new(data + i) T(a.data[i]);
        }
        // 但是程序依然有值得改进的点：
        // 1.如果对象没有默认构造,用new就错了；
        // 2.对同一片存储空间重复调用了n次构造函数，一开始没必要初始化。

        */

        // 三、完美做法：简化构造：
        data = (T *)malloc(sizeof(T) * n);
        for (int i = 0; i < n; i++) {
            new(data + i) T(a.data[i]);
        }
        return ;
```

```
    }

    T &at(int ind) { return data[ind]; } // 返回数组ind处的引用

    T &operator[](int ind) { return data[ind]; } // []的重载

    void output(int m = -1) {
        if (data == nullptr) return ;
        if (m == -1) m = n;
        cout << "arr " << this << " : ";
        for (int i = 0; i < m ; i++) {
            cout << data[i] << " ";
        }
        cout << endl;
        return ;
    }
private:
    int n;
    T *data;
};

ENDS(haizei)

BEGINS(test1)

int main() {
    haizei::Vector<int> arr1;
    for (int i = 0; i < 10; i++) arr1[i] = i; // arr.at(i) = i;
    arr1.output(10);
    haizei::Vector<int> arr2(arr1);
    arr2.output(10);
    arr2[3] = 1000;
    arr1.output(10);
    arr2.output(10);
    return 0;
}

ENDS(test1)

BEGINS(test2)

using namespace haizei;
int main() {
    Vector<A> arr1;
    for (int i = 0; i < 10; i++) {
        arr1[i].x = i;
```

```
        arr1[i].y = 2 * i;
    }
    arr1.output(10);
    Vector<A> arr2(arr1);
    arr2[3] = (A){4, 1000};
    arr1.output(10);
    arr2.output(10);
    return 0;
}

ENDS(test2)


BEGINS(test3)

using namespace haizei;
int main(){
    Vector<Vector<int>> arr1; //二维数组
    Vector<Vector<int>> arr2(arr1);
    arr2[2][2] = 1000;
    cout << "arr1========" <<endl;
    for (int i = 0; i < 3; i++) { //输出arr1前三行
        arr1[i].output(3);
    }
    cout << "arr2========" <<endl;
    for (int i = 0; i < 3; i++) { //输出arr2前三行
        arr2[i].output(3);
    }
    return 0;
}


ENDS(test3)

int main(){
    cout << "Test1:===========" << endl;
    test1::main();
    cout << "Test2:===========" << endl;
    test2::main();
    cout << "Test3:===========" << endl;
    test3::main();
    return 0;
}
```

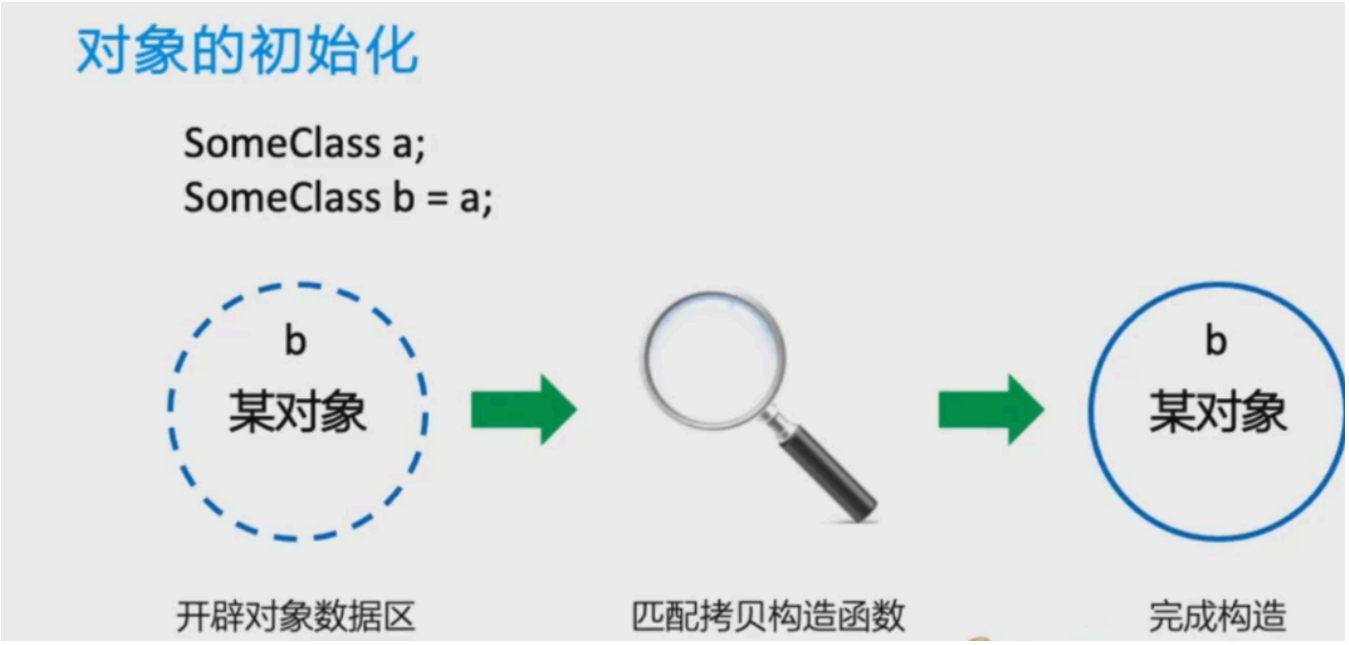


当T为`Vector<int>`的时候，data数组默认是指向空地址的。

c++ new 与malloc有什么区别 - ywliao - 博客园 (cnblogs.com)

- 我们在说一个对象的时候，往往数据区是最重要的，而且是程序中初始化完的一片数据区。
- 申请一片数据存储区的行为类似于malloc，只是有了一片需要被开垦的荒地，只有匹配了构造函数才算完成了开垦。

完整的对象——那就是一片经历了构造过程的数据存储区。

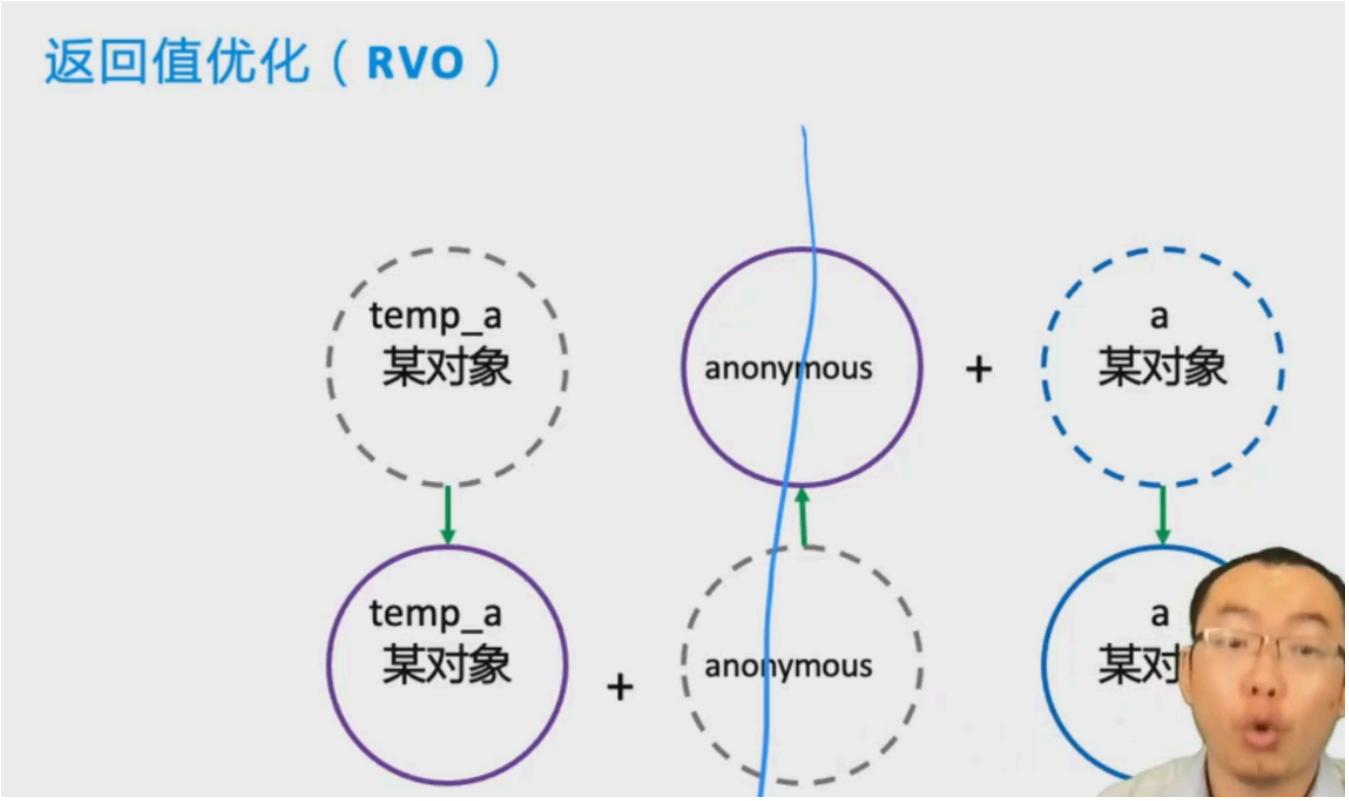


返回值优化：9.rvo.cpp

```cpp
#include<iostream>
using namespace std;
class A {
public:
    A() {
        cout << this << ": default constructor" << endl;
    }
    A(int x) {
        cout << this << ": param constructor" << endl;
    }
    A(const A &a) {
        cout << this << ": copy constructor" << endl;
    }
};

A func() {
    A temp(3);
    cout << "object temp: " << &temp << endl;
    return temp;
}

int main() {
    A a = func();
    cout << "object a: " << &a << endl;

    return 0;
}
```

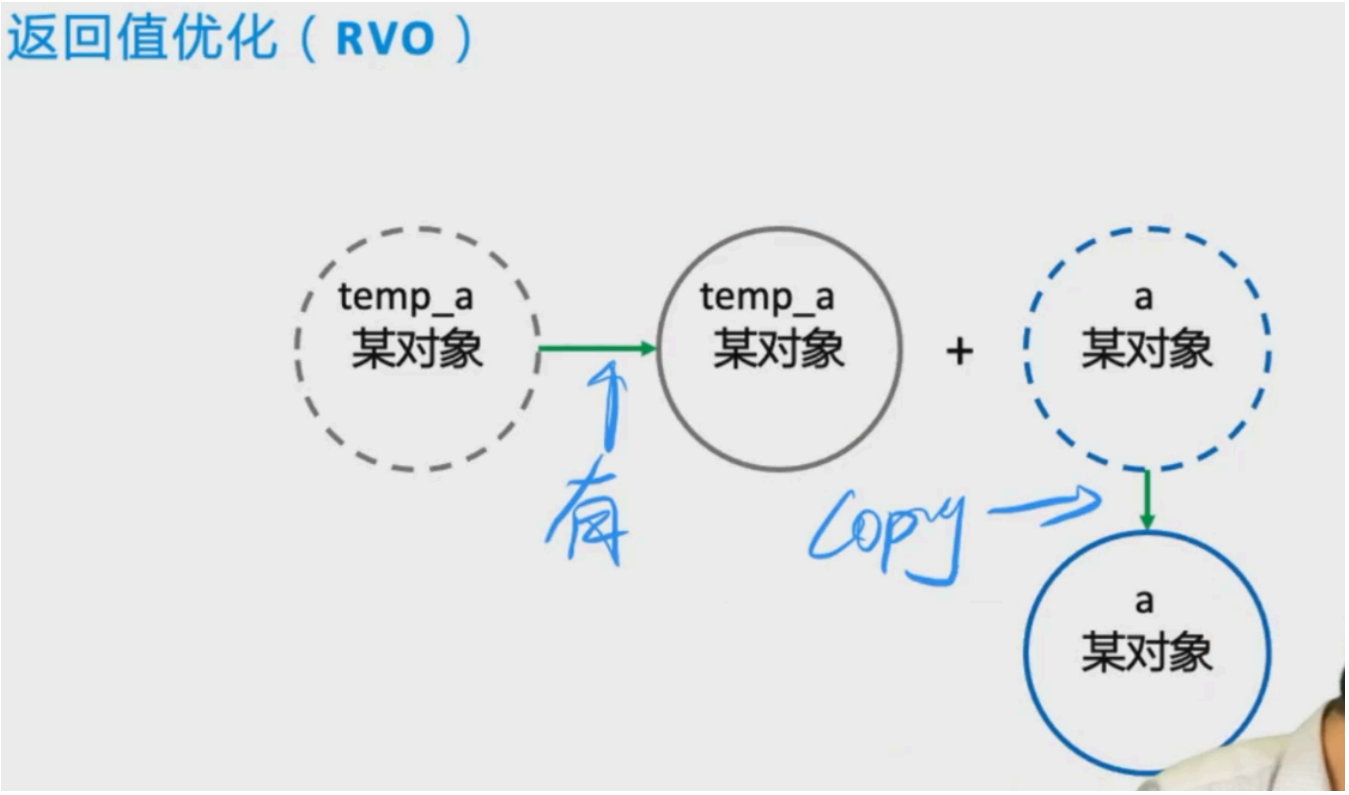
关于上面这段代码，到底调用了几次拷贝构造函数？——0次

1. **未优化版本**



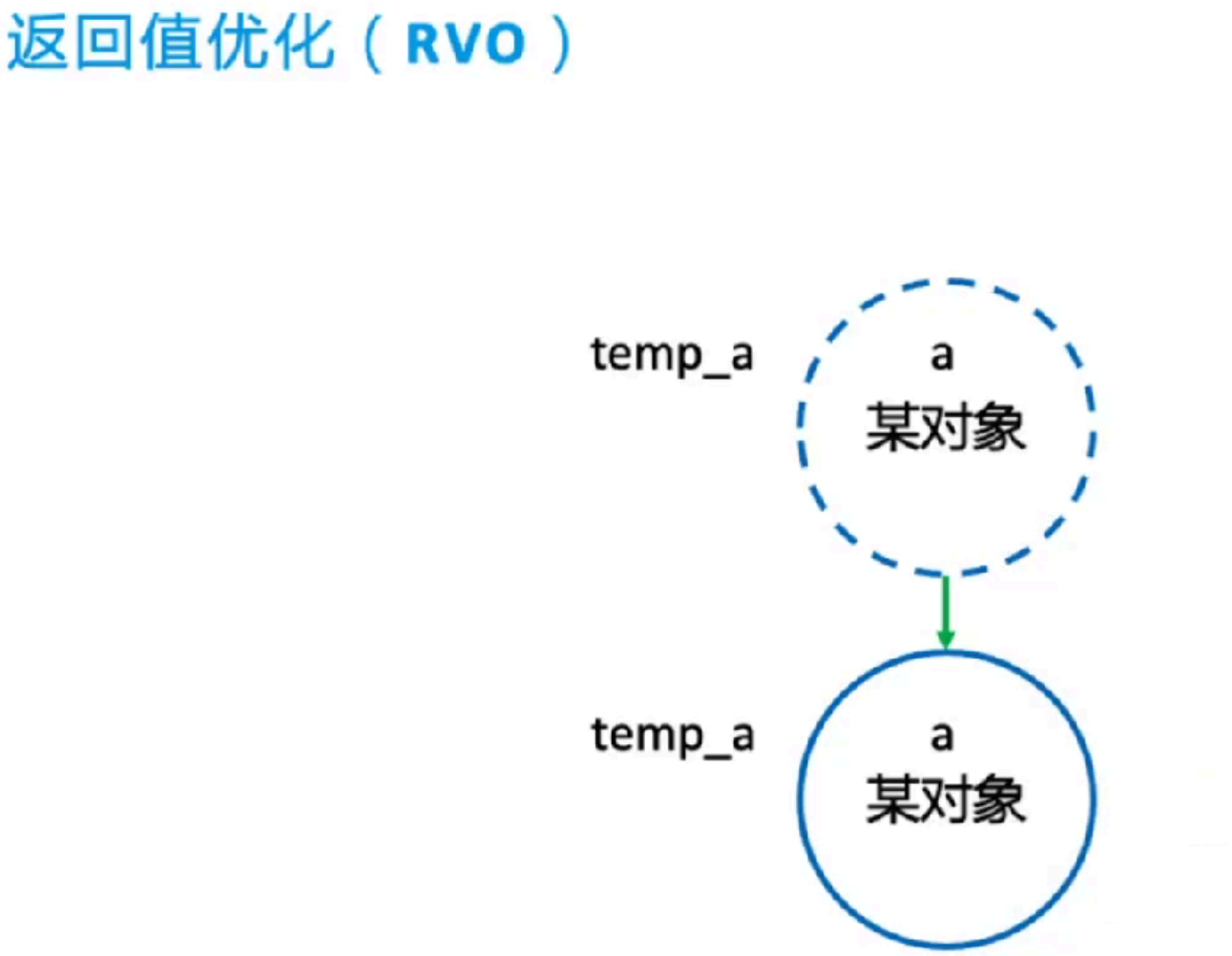
有参构造 => 拷贝构造 => 拷贝构造

2. **一次优化**



3. **进一步优化**

有参构造 => 拷贝构造




```
# 关闭返回值优化
g++ -fno-elide-constructors 9.rvo.cpp
```

关键字：10.default_delete.cpp

隐形行为显性化。我么知道他又默认的构造函数，我们是否需要使用，如果需要，就：

```
A() = default;
A() = {}
//这两行代码是没有什么差别的
```

想使用默认拷贝构造函数，就：

```
A(const A &) = default; //依次去拷贝每一个成员属性
```

注意：拷贝构造函数default和直接写{}的作用是完全不一样的。

- `A(const A &) {} //是没有依次去拷贝每一个成员属性的`

```
A(const A &) = delete; //在你想去使用相关构造函数的时候会报错
```