类的构造顺序

```
class C
{
   public:
   C()
   {
      cout << "C constructor" << endl;
   }

   ~C()
   {
      cout << "C destroy" << endl;
   }

   C(C &c)
   {
      cout << "C copy" << endl;
   }

   protected:
   private:
};
```

```
class A
{
   public:
   A()
   {
      cout << "A constructor" << endl;
   }
   ~A()
   {
      cout << "A destroy" << endl;
   }

   A(A &a)
   {
      cout << "A Copy" << endl;
   }

   virtual void Test()
   {
      cout << "Call A" << endl;
   }
   protected:

   private:

   //C c;

};


class B : public A

{

   public:

   B()

   {

      cout << ″B constructor″ << endl;

   }

   ~B()

   {

      cout << ″B destroy″ << endl;

   }

   B(B &b)

   {

      cout << ″B Copy″ << endl;

   }

   void Test()

   {
        cout << "Call B" << endl;
    }
    protected:
    private:
    C c;
};

int main(int argc, _TCHAR* argv[])
{
    B b;
    return 0;
}
```

C 在父类 A 中构造顺序为

C constructor

A constructor

B constructor

B destroy

A destroy

C destroy

C 在子类 B 中构造顺序为

```
A constructor

C constructor

B constructor

B destroy

C destroy

A destroy
```

总结：构造 父类成员 ----> 构造 父类 ------> 构造 自己成员 ----> 构造 自己

C++构造函数按下列顺序被调用：

(1)任何虚拟基类的构造函数按照它们被继承的顺序构造；

(2)任何非虚拟基类的构造函数按照它们被继承的顺序构造；

(3)任何成员对象的构造函数按照它们声明的顺序调用；

(4)类自己的构造函数。

```
#include <iostream>
using namespace std;
class OBJ1
{public:
    OBJ1(){ cout <<"OBJ1\n"; }
};
class OBJ2
```

```
{public:
    OBJ2(){ cout <<"OBJ2\n"; }
};
class Base1
{public:
    Base1(){ cout <<"Base1\n"; }
};
class Base2
{public:
    Base2(){ cout <<"Base2\n"; }
};
class Base3
{public:
    Base3(){ cout <<"Base3\n"; }
};
class Base4
{public:
    Base4(){ cout <<"Base4\n"; }
};
class Derived :public Base1, virtual public Base2,
    public Base3, virtual public Base4
{public:
    Derived() :Base4(), Base3(), Base2(),
        Base1(), obj2(), obj1()
    {
        cout <<"Derived ok.\n";
    }protected:
    OBJ1 obj1;
    OBJ2 obj2;
};
int main()
{
    Derived aa;
    cout <<"This is ok.\n";

    int i;
    cin >> i;

    return 0;
}
```

结果：

```
Base2
Base4
```

Base1

Base3

OBJ1

OBJ2

Derived ok.

This is ok.