

#include <iostream>

using namespace std;

class Base{

//将基类指针和构造方法提到了基类

//减少冗余

protected :

Base \*next;

public :

Base(Base \*t){

//使得派生类在构造时候都先调用基类构造，完成

//对下一个类的链接

next=t;

cout<<"基类的构造"<<endl;

}

virtual void fun(){

//类似递归，在链表上的对象依次调用fun方法

if(next)

next->fun();

}

virtual ~Base(){

cout<<"基类的析构"<<endl;

if(next)

delete next;

}

};

class ProductA:public Base{

public:

ProductA(Base \*t):Base(t){

cout<<"ProductA的构造"<<endl;

}

void fun(){

cout<<"ProductA的fun方法"<<endl;

Base::fun();

}

~ProductA(){

cout<<"ProductA的析构"<<endl;

}

};

class ProductB:public Base{

public:

ProductB(Base \*t):Base(t){

cout<<"ProductB的构造"<<endl;

}

void fun(){

cout<<"ProductB的fun方法"<<endl;

Base::fun();

}

~ProductB(){

cout<<"ProductB的析构"<<endl;

}

};

class Decorator:public Base{

public :

Decorator(Base \*t):Base(t){

cout<<"Decorator的构造"<<endl;

}

virtual void fun()=0;

~Decorator(){

cout<<"Decorator的析构"<<endl;

}

};

class DecoratorA:public Decorator{

public :

DecoratorA(Base \*t):Decorator(t){

cout<<"DecoratorA的构造"<<endl;

}

void fun(){

cout<<"DecoratorA的fun方法"<<endl;

Base::fun();

}

~DecoratorA(){

cout<<"DecoratorA的析构"<<endl;

}

};

class DecoratorB:public Decorator{

public :

DecoratorB(Base \*t):Decorator(t){

cout<<"DecoratorB的构造"<<endl;

}

void fun(){

cout<<"DecoratorB的fun方法"<<endl;

Base::fun();

}

~DecoratorB(){

cout<<"DecoratorB的析构"<<endl;

}

};

int main()

{

Base \*head=NULL;

head=new DecoratorB(head);

head=new DecoratorA(head);

head=new ProductA(head);

head->fun();

delete head;

}