1-5BACBB

6-10DDCCB

11-15DCDAC

16 ->

17 this指针

18 友元

19 初始化成员列表

20 delete []pa

21 不是

22 纯虚函数 纯虚函数 抽象类

23 先析构子类在析构父类

24 重载多态 包含多态 参数多态

25 不可以 可以

26 虚基类 作用域分辨符

27 static ::

28 抛出异常 处理异常

29 （1）用类的一个对象去初始化另一个对象的时候

1. 当函数的参数是类的对象时即值的传递时
2. 当函数的返回值是类的对象或者引用

30 编译时的多态是静态的 运行时的多态是动态的

编译时多态通过函数重载和运算符重载来实现，运行时的多态通过虚函数来实现。

31 优点：模块化，实体可以来被表示为类以及同一名字空间中具有相同功能的类，可以在名字空间中添加一个类而不影响该名字空间的其他成员，采用[面向对象思想](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1%E6%80%9D%E6%83%B3&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)设计的结构抄，可读性高，由于继承的存在，即使改变需求，那么维护也只是局部模块，所以维护起来是非常方便和较低成本的。代码重用功能是被封装在类中的，类是作为一个独立实体而存在的，因此可以很简单的提供类库，使代码得以重复使用。

缺点：有时会过度封装。因为类调用时需要实例化，开销比较大，比较消耗资源，性能比面向过程低。

32.

20

20

30

20

33.

Instance 1 is constructed

Instance 1 is constructed

Instance 2 is constructed

34.

i:10

j:12

35.

B's constructor called

B's default constructor called.

C's constructor called.

**5**

**6**

C's destructor called.

B's destructor called.

B's destructor called.

36.12行 类A 不存在默认构造函数

B(int i,int j,int k,int l):A(i,j){x=k;y=l;}

15行 显示的是类B的Show函数

void fl(){A::Show();}

37.

virtual 3.14\*r\*r

38.

#include<iostream>

using namespace std;

class Complex {

public:

Complex(double x = 0, double y = 0) { m\_real = x; m\_image = y; }

~Complex() {}

Complex operator+(const Complex& rhs);

Complex operator-(const Complex& rhs);

Complex operator\*(const Complex& rhs);

friend ostream& operator<<(ostream& os, const Complex& rhs);

private:

double m\_real, m\_image;

};

#include”Complex.h”

#include<iostream>

using namespace std;

Complex Complex::operator+(const Complex& rhs)

{

this->m\_real += rhs.m\_real;

this->m\_image += rhs.m\_image;

return \*this;

}

Complex Complex::operator-(const Complex& rhs)

{

this->m\_real -= rhs.m\_real;

this->m\_image -= rhs.m\_image;

return \*this;

}

Complex Complex::operator\*(const Complex& rhs)

{

this->m\_real = this->m\_real \* rhs.m\_real - this->m\_image \* rhs.m\_image;

this->m\_image = this->m\_real \* rhs.m\_image - this->m\_image \* rhs.m\_real;

return \*this;

}

ostream& operator<<(ostream& os, const Complex& rhs)

{

os << rhs.m\_real << "+" << rhs.m\_image << "i";

return os;;

LineString::LineString(Point\* pnts, int num) //构造函数，通过传入点数组来构造

{

m\_num =num ;

m\_data = new Point[m\_num];

for (int i = 0; i < m\_num; i++)

{

m\_data[i] = pnts[i];

}

}

LineString::LineString(const LineString& another)//复制构造函数

{

m\_num = another.m\_num;

m\_data = new Point[another.m\_num];

for (int i = 0; i < m\_num; i++)

{

m\_data[i] = another.m\_data[i];

}

}

LineString::~LineString()//析构函数

{

delete[] m\_data;

m\_num = 0;

}

LineString & LineString:: operator =(const LineString& rhs)//赋值函数

{

delete[]m\_data;

m\_num = rhs.m\_num;

m\_data = new Point[m\_num];

for (int i = 0; i < m\_num; i++)

{

m\_data[i] = rhs.m\_data[i];

}

return this;

}

LineString & Point:: operator[](int index)//返回线串中第index个点的引用

{ return m\_data[index];

}