面向对象结课试卷

11J192 常旭静

一

1.A 2. A 3.C 4.B 5.D 6.C 7.C 8.C 9.D 10.D 11.B 12. 13.D 14. A 15.D

二

16. \* 17.this指针 18.友元 19.构造函数 20. delete[]pa 21.不是

22.纯虚函数 纯虚函数 抽象类

23.先调用子对象的析构函数，再调用基类的析构函数

24.参数多态，包含多态，重载多态

25.不可以 可以

26.域名运算符 同名覆盖

27.class .

28.抛出异常 捕获异常

29.调用函数时产生临时变量，

30.编译时的多态是在程序写好时就已经确定的，运行时的多态是要到具体调用时才能清楚调用的具体对象。

编译时的多态通过类的继承来实现；运行时的多态可以通过虚函数和抽象类来实现。

31.优点：容易维护 有了类的继承和多态以后，在升级更新或者维护的时候代码量都比较小 代码可重用性高

不足：封装性强 数据耦合性高

三

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.

四

36. 第12行 （i，j）前没有对应的变量

第15行 AB中均含有Show函数，这样使用只能输出x y的值而非预期的a b的值

改正方法：

第12行： B(int i, int j, int k, int l):A(i, j){x=k, y=l}

第15行：void f1(){A::Show();}

37 【1】virtual

【2】3.14\*r\*r

五

38.

//Complex.h

class Complex  
{

double m\_real;//实部

double m\_im;//虚部

public:  
Comples(double a, double b)；//构造函数

Complex(Comlpex &c);//复制构造函数

~Complex(){}；//析构函数

void print();//输出

Complex operator+(const Complex& c) const;//加法运算

Complex operator-(const Complex& c) const;//减法运算

Complex operator\*(const Complex& c) const;//减法运算

friend ostream& operator<<(ostream & out, const Complex& c);

};

//Complex.cpp

#include”Complex.h”

#include<math.h>

#include<iostream>

using namespace std;

Complex ::Complex(double a ,double b)

{  
m\_real=a;

m\_im=b;}

Complex ::Complex(Comlpex &c)

{  
m\_real=c.m\_real;

m\_im=c.m\_im;

}

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

{

Complex tmp;

tmp.m\_real=m\_real+c.m\_real;

tmp.m\_im=m\_im+c.m\_im;

return tmp;

}

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

{

Complex tmp;

tmp.m\_real=m\_real-c.m\_real;

tmp.m\_im=m\_im-c.m\_im;

return tmp;

}

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

{

Complex tmp;

tmp.m\_real = (m\_real \* c.m\_real) - (m\_image \* c.m\_image);

tmp.m\_image = (m\_real \* c.m\_image) + (m\_image \* c.m\_real);

return tmp;

}

ostream&operaror<<(ostream&os,const Complex&c)

{ if(m\_im==0)

{

os<<c.m\_real<<endl;

}

if(m\_im>0)

{

os<<c.real<<”+”<<c.m\_im<<endl;

}  
if(m\_im<0)

{

os<<c.real<<”-“<<fabs(c.m\_im)<<endl;

}

if(m\_real=0)

{

os<<c.im<<endl;

}

39

//complex.h

class Point {

public:

double x, y;

};

class LineString {

Point\* m\_data;

int m\_num;

public:

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

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

~LineString(); // 析构函数

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

Point& operator[](int index);

};

//complex.cpp

#include "mycomplex.h"

LineString::LineString(Point\* pnts, int num)

{

m\_num = num;

m\_data = new Point(\*pnts);

for (int i = 0;i < num;i++,pnts++) {

m\_data[i] = \*pnts;

}

}

LineString::LineString(const LineString& another)

{

if(m\_data){  
delete[]m\_data;

}

m\_num = another.m\_num;

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

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

{

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

}

}

LineString::~LineString()

{

delete[]m\_data;

}

LineString& LineString::operator=(const LineString& rhs)

{

if(m\_data){  
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];

}

}

Point& LineString::operator[](int index)

{

return m\_data[index - 1];

}