面向对象检测卷

姓名：郭远河 班级：11J192 学号：20191000622

1-5：BDCDB

6-10:ADBDD

11-15:BCDAD

16:->

17:this指针

18：友元

19：初始化列表

20：delete[]pa

21：不是

22：纯虚函数 纯虚函数 抽象类

23：先执行派生类的，后执行基类的析构函数

24：过载多态、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： ： ：或对象名加点操作

28：抛出异常 捕获异常

29：①当用类的一个对象去初始化该类的另一个对象时。

②当函数的形参为类的对象时，调用函数的时候系统会自动调用拷贝构造函数。

③当函数返回值时类的对象时。

30：①编译时的多态由静态连编实现，运行时由动态连编实现。

②编译时由函数和运算符重载实现，运行时由虚函数实现。

31：优点：1、类的模块化使得程序易于维护。

2、类的功能使得程序灵活性高。

缺点：比其他语言而言难度较高。

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.

36：①第12行（i，j）前缺少基类的名称，构造函数有错误。

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

37、virtual

3.14\*r\*r

38：#include<iostream>

using namespace std;

class Complex {

private:

double myre;

double myim;

public:

Complex( double inputre, double inputim);

Complex operator+( Complex& theanother);

Complex operator-( Complex& theanother);

Complex operator\*( Complex& theanother);

friend ostream& operator<<(ostream& out, Complex wanttoout);

};

Complex::Complex( double inputre, double inputim)

{

myre = inputre; myim = inputim;

}

Complex Complex:: operator+( Complex& theanother) {

myim += theanother.myim;

myre += theanother.myre;

Complex mid(myre,myim);

return mid;

}

Complex Complex::operator-( Complex& theanother) {

myim -= theanother.myim;

myre -= theanother.myre;

Complex mid(myre, myim);

return mid;

}

Complex Complex::operator\*( Complex& theanother) {

myre = (myre \* theanother.myre) - (myim \* theanother.myim);

myim = (myre \* theanother.myim) + (myim \* theanother.myre);

Complex mid(myre, myim);

return mid;

}

ostream& operator<<(ostream& out, Complex wanttoout) {

out << "实数的实部 " << wanttoout.myre << " " << "实数的虚部" << wanttoout.myim << "i";

return out;

}

int main() {

Complex c1(1.0, 2.0);

Complex c2(1.5, 25.0);

cout << c1 + c2 << endl;

cout << c1 - c2 << endl;

cout << c1 \* c2 << endl;

}

39：#include<iostream>

using namespace std;

class Point {

public:

double x;

double y;

};

class LineString {

public:

LineString(Point\* pnts, int num);

LineString(const LineString& another);

~LineString();

LineString& operator =(const LineString& rhs);

Point& operator[](int index);

private:

Point\* m\_data;

int m\_num;

};

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

m\_num = num;

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

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

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

}

}

LineString::LineString(const LineString& another) {

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;

m\_data = nullptr;

m\_num = 0;

}

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

if (this != &rhs) {

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];

}

}

else

return \*this;

}

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

return m\_data[index-1];

}