面向对象检测卷

姓名：胡宇昊

学号：20191003167

班级：11j192

一、

1-5 B D C B B

6-10 A D B D D

11-15 B C D A D

二、

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. 先执行派生类的析构函数，再执行基类的析构函数

24. 过载多态、参数多态、包含多态

25. 不可以 可以

26. 虚基类 作用域分辨符

27. private 对象名加点操作符

28. 抛出异常、传递信息

捕获异常

29.答：

（1）当用类的一个对象初始化该类的另一个对象时。

point A(1,2);

point B(A);

（2）如果函数的形参是类的对象,调用函数时,进行形参和实参结合时。

（3） 如果函数的返回值是类的对象,函数执行完成返回调用者时。

point g()

{

point A(1,2);

return A;

}

void main()

{

point B;

B = g();

}

（4）需要产生一个临时类对象时。

30.答：

（1）不同：

编译时的多态是通过静态连编来实现的；

运行时的多态是用动态连编来实现的。

（1）实现：

编译时的多态性主要是通过函数重载和运算符重载来实现的；

运行时的多态性主要是通过虚函数来实现的。

31.答：

优点：

（1）质量高；在设计时，可重用现有的，在以前的项目的

领域中已被测试过的类使系统满足业务需求并具有较高的质量。

（2）效率高；

（3）易扩展；由于继承、封装、多态的特性，使得系统更灵活、

更容易扩展，而且成本较低。

缺点:

（1）没有准确的定义

（2）维护困难

（3）不适合所有的应用当用于普通应用时，其性能会降低

并要求很高的处理能力。

三、

32.答：

20

20

30

20

33.答：

Instance 1 is constructed

Instance 2 is constructed

Instance 3 is constructed

34.答：

i:10

j:12

35.答：

B's construcyor called.

B's default constructor called.

C's constuctor called.

5

6

C's destructor called.

B's destructor called.

B's destructor called.

36.答：

错误1:

15行，显示的是x、y而非a、b

修改为：

void f1（）{A::Show();}

错误2：

12行，B的构造函数的初始化列表

修改为：

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

37.答：

【1】：virtual

【2】：3.14\*r\*r

38.答：

#include<iostream>

using namespace std;

class Complex {

private:

double m\_true;

double m\_false;

public:

Complex(double a, double b) :m\_true(a), m\_false(b) {}

~Complex(){}

const Complex operator+(const Complex &a) {

this->m\_true += a.m\_true;

this->m\_false += a.m\_false;

Complex \*m(this);

return \*m;

}

const Complex operator-(Complex& a) {

this->m\_true -= a.m\_true;

this->m\_false -= a.m\_false;

Complex\* m(this);

return \*m;

}

const Complex operator\*(Complex& a) {

double f, t;

t = this->m\_true \* a.m\_true - this->m\_false \* a.m\_false;

this->m\_true = t;

f = this->m\_true \* a.m\_false + this->m\_false \* a.m\_true;

Complex\* m(this);

return \*m;

}

friend ostream& operator<<(ostream& b, Complex a);

};

ostream& operator<<( ostream &b,Complex a) {

cout << a.m\_true;

if (a.m\_false > 0)

cout << "+" << a.m\_false << "i" << endl;

if (a.m\_false < 0)

cout << a.m\_false << "i" << endl;

if (a.m\_false == 0)

cout << endl;

return b;

}

void main() {

Complex c1(2.0, 3.0);

Complex c2(1.0, 2.0);

cout << c1 + c2 << endl;

cout << c1 - c2 << endl;

cout << c1 \* c2 << endl;

}

39.答：

#include<iostream>

using namespace std;

class Point {

public:

double x, y;

friend class LineString;

};

class LineString { //由多个点组成的线串类，由 n 个点组成的折线包含 n-1 个线段

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) {

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

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

LineString \*m(this);

return \*m;

}; // 赋值函数

Point& operator[](int index) {

return m\_data[index];

}//返回线串中第 index 个点的引用

private:

Point \*m\_data; // 用于保存组成线串的点

int m\_num;// 线串中点的数量

};

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

\*m\_data = \*pnts;

m\_num = num;

}

LineString::~LineString() {

m\_num = 0;

}

LineString::LineString(const LineString& another) {

m\_data = another.m\_data;

m\_num = another.m\_num;

}