**面向对象程序设计检测卷**

姓名：李坤 学号：20191000465

选择题：

1~5：BDCBB 6~10: ADBDD 11~15:BCDAD

填空题：

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. 先执行子类的析构函数，再执行父类类的析构函数  
  
24. 参数多态、过载多态、包含多态  
  
25. 不可以  可以  
  
26. 虚基类   作用域分辨符  
  
27. 对象名加点操作符 private  
  
28. 抛出异常、传递信息     
      捕获异常  
  
29.答：

（1） 如果函数的返回值是这个类的对象,函数执行完成后返回到调用者的时候。  
Line：： get()  
{

Line c（5）；

Return c；  
   }  
void main()  
{

Line L1;

L1=get（）；  
}

（2）当用类的一个对象初始化该类的另一个对象时。  
   Line line1(1);Line line2(line1);

（3）如果函数的形参是这个类的对象,那么调用函数时,当进行形参和实参结合时就会拷贝构造。  
  
（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）重复性强：在设计时，可重用现有的代码，或者是使用以前开发过的  
领域中已被测试过的类。  
（2）易扩展；由于继承、封装、多态的特性，能够设计出高  
内聚、低耦合的系统结构，使得系统更加灵活、更容易扩展，  
而且成本较低。  
缺点:  
（1）维护困难：随着组织信息需求的改变，对象的定义也要求  
改变并且需要移植现有数据库，以完成新对象的定义。当改变对象  
的定义和移植数据库时，它可能面临真正的挑战。  
（2）对初学者来说难度过大。

程序阅读：

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:

在12行（i，j）前面缺少基类的名称。类A不存在相应的构造函数。

建议修改为B(int i, int j,int k,int l):A(i, j) { x =k; y = l; } //构造函数

在第15行注释说显示ab的值，但调用B的show函数显示x，y的值

建议修改为 void f1( )A::{Show( );}

37:

1. virtual
2. 3.14\*r\*r;

编程题：

38:

#include<iostream>

using namespace std;

class Complex {

private:

double re;

double im;

public:

Complex(const double x, const double y);

Complex(const Complex\* d);

Complex operator+(const Complex& a);

Complex operator-(const Complex& b);

Complex operator\*(const Complex& c);

friend ostream& operator<<(ostream& out, Complex c1);

};

Complex::Complex(const double x, const double y):

re(x),im(y)

{}

Complex::Complex(const Complex\* d) {

this->im = d->im;

this->re = d->re;

}

Complex Complex:: operator+(const Complex& a) {

this->im += a.im;

this->re += a.re;

Complex newa(this);

return newa;

}

Complex Complex::operator-(const Complex& b) {

this->im -= b.im;

this->re -= b.re;

Complex newb(this);

return newb;

}

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

this->re = (this->re \* c.re) - (this->im \* c.im);

this->im = (this->re \* c.im) + (this->im \* c.re);

Complex newc(this);

return newc;}

ostream& operator<<(ostream& out, Complex c1) {

out << "实部 " << c1.re << " " << "虚部" << c1.im << "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\_data = new Point[num];

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

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

}

m\_num = num;

}

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) {

return \*this;

}

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];

}

delete[]rhs.m\_data;

}

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

return m\_data[index];

}