一、单项选择题

B D C B B

C D B D D

B C D A D

二、填空与简答题

16.->

17.this指针

18.友元

19.初始化列表

20.delete[] pa

21.不是

22.纯虚，纯虚函数，抽象类

23.先执行派生类的析构函数，然后再去执行基类的析构函数。

24.重载多态、类型参数化多态、包含多态

25.不可以，可以

26.虚基类，作用域解析运算符

27.private 点.

28.抛出异常 捕获任何类型的异常

29. ①用一个类的对象对另一个新对象进行初始化时。②函数参数为类的对象。③.函数的返回值是类的对象。

30. 静态联编支持的多态性称为编译时多态性，动态联编所支持的多态性称为运行时多态性。编译时多态性是通过函数重载（包括运算符重载）和模板实现的。运行时多态性是通过虚函数来实现的。

31. 优点①易于维护，可读性高②可重用，减少工作量，减少程序出错③易扩展，继承，多态等特性使得程序更加灵活

缺点：程序处理效率相比面向过程更低。

三、看程序写结果

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、综合题

36. 指出下面程序类 B 成员函数中的两处错误，并提出改正方法.

①12行（i,j)处，初始化之前没写对象，应改为 B(int i,int j,int k,int l):A(i,j){x=k;y=l;}

②15行 在类外使用show()成员函数时没有标出对象名，应改为 void f1( ){A::Show( );}

37．程序填空

【1】virtual

【2】3.1415926\*r\*r

1. 头文件Complex.h：

class Complex

{

public:

Complex() { //默认构造函数

\_real = 0;

\_image = 0;

}

Complex(double real , double image ){ //构造函数

\_real = real;

\_image = image;

}

Complex(const Complex& com){ //拷贝构造函数

\_real = com.\_real;

\_image = com.\_image;

}

~Complex() {} //析构函数

Complex operator+ (const Complex& d){ //+重载

Complex com;

com.\_real = (this->\_real + d.\_real);

com.\_image = (this->\_image + d.\_image);

return com;

}

Complex operator- (const Complex& d){ //-重载

Complex com;

com.\_real = (this->\_real - d.\_real);

com.\_image = (this->\_image - d.\_image);

return com;

}

Complex operator\* (const Complex& d) { //\*重载

Complex com;

com.\_real = (this->\_real \* d.\_real)-(this->\_image \* d.\_image);

com.\_image = (this->\_image \* d.\_real) + (this->\_real \* d.\_image);

return com;

}

friend ostream& operator<< (ostream& os,const Complex& d) { //<<重载

if (d.\_image > 0) {

os << d.\_real << "+" << d.\_image << "i" << endl;

return os;

}

else {

os << d.\_real << d.\_image << "i" << endl;

return os;

}

}

private:

double \_real;

double \_image;

};

39.

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

this->m\_data = pnts;

this->m\_num = num;

}

LineString::LineString(const LineString& another) {

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

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

}

LineString::~LineString() {};

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

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

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

return \*this;

}

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

return this->m\_data[index];

}