电子科技大学信息与软件工程学院

**课外作业**

点名册序号 304-3

学 号 2015220103022

姓 名 张健顺

课程名称 面向对象程序设计(C++)

理论教师 陈安龙

1. **作业序号:第1次**
2. **作业题目与要求：**

定义一个圆类，数据成员包括圆心坐标，半径，成员方法包括：构造函数，析构函数，计算圆面积、周长，显示圆心坐标、半径。

1. **数据结构与解答思路：**

#include<iostream>

using namespace std;

class Circle

{

private:

double radius;

double central\_X, central\_Y;

public:

Circle(double r,double x,double y)

{

radius = r; central\_X = x;central\_Y = y;

}

~Circle(){}

double Get\_girth(){return 3.14 \* 2 \* radius;}

double Get\_area(){return 3.14\*radius\*radius;}

void show()

{

cout << "radius=" << radius << endl;

cout << "central\_X=" << central\_X << ",central\_Y=" << central\_Y << endl;

}

};

1. **程序代码：**

#include<iostream>

using namespace std;

class Circle

{

private:

double radius;double central\_X, central\_Y;

public:

Circle(double r,double x,double y){

radius = r; central\_X = x;central\_Y = y;

}

~Circle(){}

double Get\_girth(){return 3.14 \* 2 \* radius;}

double Get\_area(){return 3.14\*radius\*radius;}

void show(){

cout << "radius=" << radius << endl;

cout << "central\_X=" << central\_X << ",central\_Y=" << central\_Y << endl;

}

};

1. **结果分析与体会：**

int main()

{

/\*Circle cr(10,1,1); //The data is set in advance!\*/

double r;

double x, y;

cout << "Please enter the radius:";

cin >> r;

cin.get();

cout << "Please enter the coordinate:";

cin >> x >> y;

cin.get();

Circle cr(r, x, y);

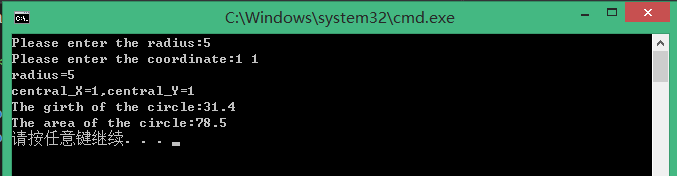
cr.show();

cout << "The girth of the circle:" << cr.Get\_girth() << endl;

cout << "The area of the circle:" << cr.Get\_area() << endl;

return 0;

}



自定义输入圆半径以及圆心坐标，打印出圆心坐标，分别调用相应的计算周长和计算面积的方法打印出相应的面积和周长。

构造函数和析构函数中可以自定义输出相应语句，从而便于理解对象的创建与销毁过程。

1. **作业序号：第2次**
2. **作业题目与要求：**

定义一个日期类，数据成员包括年、月、日、小时、分、秒，成员方法包括：

– 打印显示日期对象年月日、小时、分秒；

• 格式如：2013-09-15:14:50;

– 实现秒、分、小时、日、月、年的增减，

– 如果创建日期对象时，小时、分、秒缺省为

00:00:00,日期缺省为2000-01-01:00:00:00

– 定义能实现日期对象复制的拷贝构造函数

1. **数据结构与解答思路：**

class Time

{

public:

Time(int y = 2001, int M = 1, int d = 1, int h = 0, int m = 0, int s = 0);

void Print();

Time(const Time& t);

Time& Modify(Time& time, int y = 0, int M = 0, int d = 0, int h = 0, int m = 0, int s = 0);

void askUsers(Time& t);

private:

int year,month,day,hour, minute,second;

};

构造函数中使用默认参数，通过用户自定义的方式来修改时间。

1. **程序代码：**

**Time.cpp**

#include"time.h"

using namespace std;

Time::Time(int y, int M, int d, int h, int m, int s){

year = y;month = M;day = d;hour = h;minute = m;second = s;

}

void Time::Print(){

cout << "当前时间为：";

cout << setw(4) << setfill('0') << year << "-";

cout << setw(2) << setfill('0') << month << "-";

cout << setw(2) << setfill('0') << day << ":";

cout << setw(2) << setfill('0') << hour << ":";

cout << setw(2) << setfill('0') << minute << ":";

cout << setw(2) << setfill('0') << second << endl;

}

**Time.h**

class Time

{

public:

Time(int y = 2001, int M = 1, int d = 1, int h = 0, int m = 0, int s = 0);

void Print();

Time(const Time& t);

Time& Modify(Time& time, int y = 0, int M = 0, int d = 0, int h = 0, int m = 0, int s = 0);

void askUsers(Time& t);

private:

int year,month,day,hour, minute,second;

};

Time::Time(const Time& t)

:year(t.year), month(t.month), day(t.day), hour(t.hour), minute(t.minute), second(t.second){cout << "复制构造函数执行了ｙｅａｈ！" << endl;}

Time& Time::Modify(Time& time, int y, int M, int d, int h, int m, int s){

year += y;

if ((month + M) != 12) month = (M + month) % 12;

if ((day + d) != 30) day = (day + d) % 30;

hour = (hour + h) % 24; minute = (minute + m) % 60;

second = (second + s) % 60; return time;

}

void Time::askUsers(Time& t){

cout << "Do you want to modify the time? y or n" << endl;

char ch1; cin >> ch1;

if (ch1 == 'n') exit(0);

else{

cout << "Which one do you want to modify?( y,M,d,h,m,s )" << endl;

char ch2; cin >> ch2;

cout << "Add:please enter a number( >0 )" << endl;

cout << "Minus:Please enter a number( <0 )" << endl;

int a; cin >> a;

switch (ch2){

case 'y':t.Modify(t, a); break;

case 'M':t.Modify(t, 0, a);break;

case 'd':t.Modify(t, 0, 0, a); break;

case 'h':t.Modify(t, 0, 0, 0, a);break;

case 'm':t.Modify(t, 0, 0, 0, 0, a);break;

case 's':t.Modify(t, 0, 0, 0, 0, 0, a);break;

default:cout << "Wrong input !!!" << endl;

}

t.Print();

}

}

1. **结果分析与体会：**

**Main.cpp**

#include<iostream>

#include"time.h"

int main(){

Time time1(2002);time1.Print();

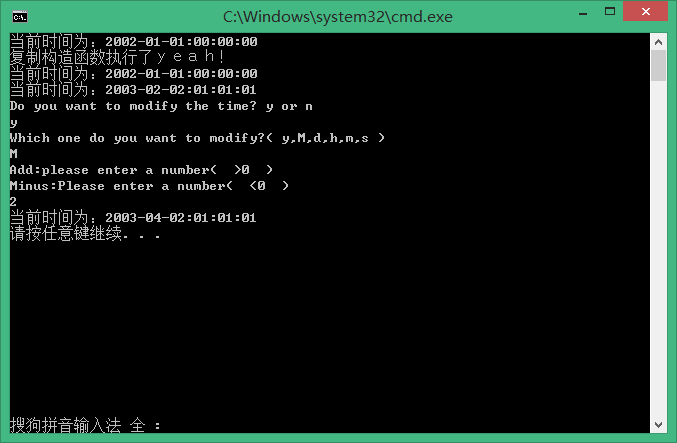
Time time2(time1);time2.Print();

time1 = time1.Modify(time1,1, 13, 1, 1, 1, 1);time1.Print();

time1.askUsers(time1);

return 0;

}



构造函数中使用默认参数，通过用户自定义的方式来修改时间。

其中还对数据做了相应的处理。譬如一年只有12月份，一个月只有三十天，所以利用到循环计数，以及闰年闰月的考虑，若要减小时间，则输入对应项的负值。同时还考虑到不能出现月份和日期不能为0的注意事项。

复制构造函数的执行过程也有了相应的体会。其中参数中调用的是类对象的常量引用。

**一、作业序号:第3次**

**二、作业题目与要求：**

定义矩形Rectangle类，包括长length、宽width，构造函数、复制构造函数、并分别用类成员函数和友元函数重载=、+、-、++、--、+=、-=、==、>、<运算符，a=b表示将矩形b赋给a；a+b表示矩形面积相加，a-b表示矩形面积的差；运算a+=b的含义为a=a+b,运算a-=b的含义为a=a-b；a>b表示a的面积大于b的面积时为1，否则为0；a<b表示a的面积小于b的面积时为1，否则为0； a==b表a的面积等于b的面积时为1，否则为0；++表示矩形的长宽实都加1，--表示矩形长宽都减1，同时要考虑前++和后++、前--和后--。分析分别在什么情况下使用复制构造函数和重载运算符=。

1. **数据结构与解答思路：**

class Rectangle

{

public:

Rectangle(int l = 5, int w = 5);

Rectangle(const Rectangle& r);

int area();void Print();

//成员函数重载

int operator +(Rectangle r);

int operator -(Rectangle r);

Rectangle& operator ++();//前缀，返回对象本身的引用

Rectangle operator ++(int);//返回临时值

Rectangle& operator --();//前缀，返回对象本身的引用

Rectangle operator --(int);//返回临时值

//参数为右值的常量引用，返回值为对左值对象的引用

Rectangle& operator =(const Rectangle& r);

**四、程序代码：**

bool operator >(Rectangle r);//只为类重载>和<两个运算符之一

friend int operator +=(Rectangle r1, Rectangle r2);

friend int operator -=(Rectangle r1, Rectangle r2);

bool operator ==(const Rectangle& r);

private:

int length;int width;

};

**Rectangle.h**

class Rectangle

{

public:

Rectangle(int l = 5, int w = 5);

Rectangle(const Rectangle& r);

int area();void Print();

int operator +(Rectangle r);

int operator -(Rectangle r);

Rectangle& operator ++();//前缀，返回对象本身的引用

Rectangle operator ++(int);//返回临时值

Rectangle& operator --();//前缀，返回对象本身的引用

Rectangle operator --(int);//返回临时值

//参数为右值的常量引用，返回值为对左值对象的引用

Rectangle& operator =(const Rectangle& r);

bool operator >(Rectangle r);//只为类重载>和<两个运算符之一

friend int operator +=(Rectangle r1, Rectangle r2);

friend int operator -=(Rectangle r1, Rectangle r2);

bool operator ==(const Rectangle& r);

private:

int length;int width;

};

**Rectangle.cpp**

#include<iostream>

#include"Rectangle.h"

using namespace std;

Rectangle::Rectangle(int l, int w){length = l;width = w;}

Rectangle::Rectangle(const Rectangle& r) :length(r.length), width(r.width)

{cout << "复制构造函数执行了" << endl;}

int Rectangle::area(){ return length\*width;}

void Rectangle::Print(){

cout << "length:" << length << " " << "width:" << width << endl;

}

int Rectangle::operator +(Rectangle r){return this->area() + r.area();}

int Rectangle::operator -(Rectangle r){return area() - r.area();}

int operator +=(Rectangle r1,Rectangle r2){return r1+r2;}

int operator -=(Rectangle r1, Rectangle r2){return r1 - r2;}

Rectangle& Rectangle::operator =(const Rectangle& r){

length = r.length; width = r.width; return \*this;

}

bool Rectangle::operator >(Rectangle r){

if (this->area() > r.area())return 1;else return 0;

}

Rectangle& Rectangle::operator ++(){++length;++width;return \*this;}

Rectangle& Rectangle::operator --(){--length;--width;return \*this;}

Rectangle Rectangle::operator ++(int){

Rectangle temp(this->length, this->width);

length++;width++;return temp;

}

Rectangle Rectangle::operator --(int){

Rectangle temp(this->length, this->width);

length--;width--;return temp;

}

bool Rectangle::operator ==(const Rectangle& r)

{

if (length == r.length && width == r.width)

return 1;

else

return 0;

}

**Main.cpp**

#include<iostream>

#include"Rectangle.h"

using namespace std;

int main(){

Rectangle r1;Rectangle r2;Rectangle a(0, 0);Rectangle b(0, 0);

cout << "+ :"<< r1 + r2 << endl;

cout << "- :"<< r1 - r2 << endl;

cout << "+= :" << (r1 += r2)<< endl;

cout << "-= :" << (r1 -= r2)<< endl;

cout <<"r1 > r2 ? "<< (r1 > r2)<< endl;

cout <<"r1 == r2 ? "<< (r1 == r2)<< endl;

cout << "++"<<endl;cout << "r1 :";r1.Print();

a = r1++;cout << "r1 :";r1.Print();

cout << "a=r1++ ";a.Print();

++r1;cout << "r1 :";r1.Print();

cout << "a=++r1 ";a.Print();

cout << "--"<<endl;cout << "r2 :";r2.Print();

b=r2--;cout << "r2 :";r2.Print();

cout << "b=r2-- ";b.Print();--r2;

cout << "r2 :";r2.Print();cout << "b=--r2 ";b.Print();

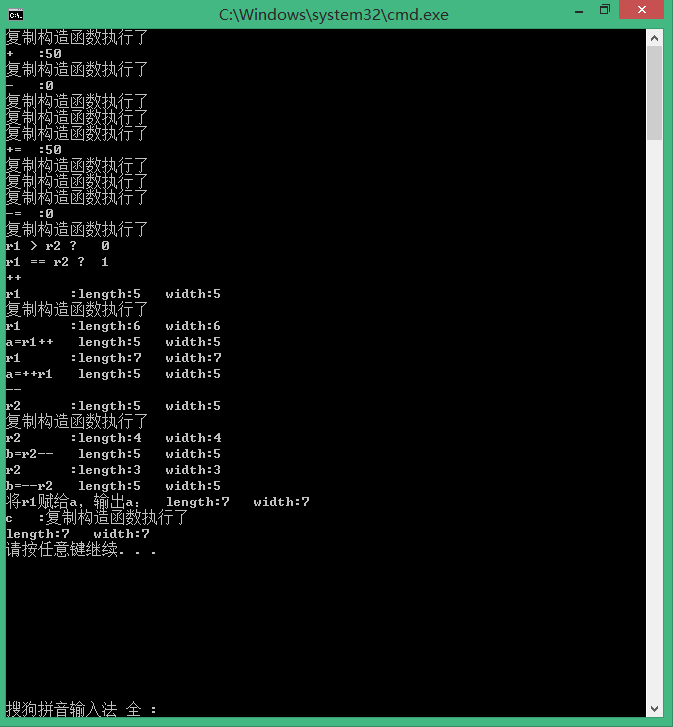
cout << "将r1赋给a，输出a： ";a = r1;a.Print();

cout << "c :";Rectangle c(r1); c.Print();

return 0;

}

**六、结果分析与体会：**



（1）运算符重载的成员函数以及友元重载的成员函数的区别有：

A.运算符重载的成员函数参数个数为原操作数个数减一，而友元重载的成员函数的参数个数为原操作数个数；

B.成员函数有this指针，友元函数无this指针；

C.成员函数运算的结果可以作为左值使用，不是一个新产生的结果，而友元重载的运算结果是一个新产生的值，并且这个值不能作为左值使用。

1. 对于运算符“=，（），[]，->”只能作为成员函数；
2. 重载函数：通过函数的参数表唯一标识并且来区分函数的一种特殊的函数。

**一、作业序号:第4次**

**二、作业题目与要求：**

在类List类内部建立一个内部类Node，有成员数据 int data，成员数据 Node \*next和Node\* pre 指针。List类具有成员变量Node \* first和 Node \* last，List类具有成员函数插入、删除结点、查找结点；两个链表的连接、结点数据的排序、两个有序链表的合并成有序链表等成员方法；构造函数、深复制构造函数、析构函数、链表的置空等。

1. **数据结构与解答思路：**

class list{

private:

class Node{

public:

int data;

Node \*head；Node \*tail;

Node();Node(int);Node(const Node\*);

~Node() {}

};Node \*first;Node \*last;

public:

list();list(const list& s);~list();

void push\_back(int);void insert(int,int);void list\_delete(int);

list::Node\* find(int);void list\_linked(list);void list\_sort();

void list\_empty();void sort\_list\_linked(list l);

void show\_node();

};

**四、程序代码：**

**list.h**

class list{

private:

class Node{

public:

int data;

Node \*pre；Node \*next;

Node();Node(int);Node(const Node\*);~Node() {}

};Node \*head;Node \*tail;

public:

list():head(NULL),tail(NULL){}

list(const list& s);~list();

void push\_back(int);void insert(int,int);void list\_delete(int);

list::Node\* find(int);void list\_linked(list);void list\_sort();

void list\_empty();void sort\_list\_linked(list l);

void show\_node();

};

list::list(const list& L){

head=L.head; tail=L.tail;

}

list::Node::Node(){data = 0; head = NULL; tail = NULL;}

list::Node::Node(int a){data = a;tail = NULL;head = NULL;}

void list::list\_empty(){

Node \*p = head, \*q;

if (p == NULL){cout << "it's null" << endl;return; }

while (p != NULL){q = p; p = p->next;delete q;}

head = tail = NULL;

}

void list::push\_back(const int& \_data){

Node \*p = new Node(\_data); p->next=NULL;

if(tail==NULL) head=tail=p;

else{ tail->next=p; tail=p; }

return \*this;

}

void list::show\_node(){

Node \*p = head;

if (first == NULL)cout << "NULL !" << endl;

else while (p != NULL){cout << p->data << " "; p = p->next;}

cout << endl;

}

void list::insert(int data, int \_data){

Node \*p = new Node(data);Node \*q = find(\_data);

if (q != first){

p->pre = q->pre; q->pre->next = p;

p->next = q; q->pre = p;

}

else{

q->pre = p; p->next = q; head = p;

}

}

void list::list\_delete(int data){

Node \*L = new Node(); L = find(data);

if (L == head) head = L->next;

else if (L == tail){tail = L->pre; tail->next = NULL;}

else{

L->pre->next = L->next;

L->next->pre = L->pre;

}

}

void list::list\_linked(list l){

if (l.head == NULL) return;

tail->next = l.head; l.head->pre = tail; tail = l.tail;

}

void list::list\_sort(){

Node \*p = head, \*q; int temp;

while (p != NULL){

q = head;

while (q != tail){

if ((q->data) > (q->next->data)){

temp = q->next->data;

q->next->data = q->data;

q->data = temp;

}

q = q->next;

}

p = p->next;

}

}

void list::sort\_list\_linked(list l)

{

list\_linked(l);

list\_sort();

}

list::Node\* list::find(int a){

Node \*p = head;

while (p != NULL){

if (p->data == a) break;

else p = p->head;

}

return p;

}

List.cpp

#include "list.h"

int main(){

list L1;cout << "created L1：";

L1.push\_back(11);L1.push\_back(99);L1.push\_back(44);

L1.show\_node();

cout << "insert 66 before 99:";

L1.insert(66, 99);

L1.show\_node();

cout << "deleted 99:" ;

L1.list\_delete(99);

L1.show\_node();

list L2;cout << "created L2:" ;

L2.push\_back(22);L2.push\_back(77);L2.show\_node();

L1.list\_linked(L2);cout << "linked L1 and L2:" ;

L1.show\_node();

cout << "insert 55 before 22 in L1:" ;

L1.insert(55, 22);

L1.show\_node();

cout << "sort L1:" ;

L1.list\_sort();

L1.show\_node();

cout << "empty L2:" ;

L2.list\_empty();L2.show\_node();

cout << "add 99 to L2:" ;

L2.push\_back(99);L2.show\_node();

cout << "linked sorted L1 and L2:";

L1.sort\_list\_linked(L2);L1.show\_node();

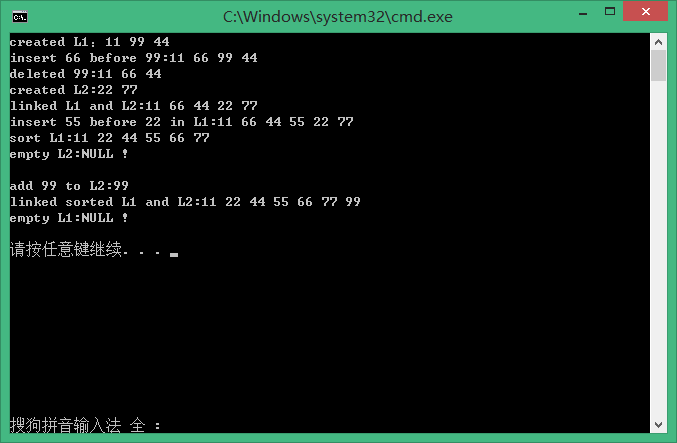
cout << "empty L1:";

L1.list\_empty();L1.show\_node();

return 0;

}

1. **结果分析与体会：**



**实验结果分析：**

首先创建链表L1；

在L1中的某个元素之前出入新的元素；

删除L1其中一个元素；

创建链表L2；

将L2和L1拼接起来，即在L1末尾添加L2；

合并的链表为L1；

在L1中指定元素前插入元素；

对合并的链表进行排序；

将L2置空；

再向L2中添加元素；

和之前的链表进行保序合并；

将L1置空。

**体会：**

结点类在链表类中声明，即为内部类，同样有相应的构造函数对结点进行初始化。

链表的创建其实可以通过简单的尾插法来实现。创建完成以后对链表的增删该查方法，都涉及到链表中要处理的位置的结点的指针指向。

针对结点中的数据域的数据类型可能不同，可以设计模板类来编写，甚至使用到容器类。同时程序中多次使用到结点的遍历，可以采用迭代器进行相关操作，进行代码重构。

**一、作业序号:第5次**

**二、作业题目与要求：**

阅读下列程序 1、程序 2、程序 3，写出输出结果，并上机调试这三个程序，分析

说明它们输出不同的原因，并总结出输出特点和规律。

**三、程序代码：**

程序1中两个基类的声明：

class Graphics{

public:

virtual double area() = 0;

virtual void draw() { cout << "Graphics的draw()被调用" << endl; };

virtual ~Graphics(){ cout << "Graphics类对象被析构" << endl << endl;}

};

class Points : public Graphics{

protected:

int x; int y;

public:

Points() :x(0), y(0){};

Points(int x0, int y0) :x(x0), y(y0){}; double area(){return 0;};

void draw(){cout << "Points的draw()被调用" << endl; cout << "x=" << x << ",y=" << y << endl;

}

~Points(){ cout << "Points类对象被析构" << endl; }

};

class Circle : public Points{

private:

int r;

public:

Circle() :Points(), r(0){};

Circle(int x0, int y0, int r0) :Points(x0, y0), r(r0){};

};

double area(){return 3.14\*r\*r;};

void draw(){

cout << "Circle的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",r=" << r << endl;

cout << "Circle的面积为：" << area() << endl;

}

~Circle(){ cout << "Circle类对象被析构" << endl; }

};

class Ellipse : public Points{

private:

int a;

int b;

public:

Ellipse() :Points(), a(0), b(0){};

Ellipse(int x0, int y0, int a0, int b0) :Points(x0, y0), a(a0), b(b0){};

double area(){ return 3.14\*a\*b; };

void draw(){

cout << "Ellipse的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",a=" << a << ",b=" << b << endl;

cout << "Ellipse的面积为：" << area() << endl;

}

~Ellipse(){ cout << "Ellipse类对象被析构" << endl; }

};

int main(){

Graphics \*ptr;

cout << "Graphics类指针指向派生类：" << endl << endl;

ptr = new Points(1, 1);ptr->draw();delete ptr;

ptr = new Circle(2, 2, 4); ptr->draw();delete ptr;

ptr = new Ellipse(3, 3, 2, 1);ptr->draw();delete ptr;

Points \*p\_ptr;cout << "Points类指针指向派生类：" << endl << endl;

p\_ptr = new Circle(4, 4, 8);p\_ptr->draw(); delete p\_ptr;

p\_ptr = new Ellipse(6, 6, 4, 2);p\_ptr->draw();delete p\_ptr;

return 0;

}

程序2

#include <iostream>

#include <string.h>

using namespace std;

class Graphics{

public:

virtual double area() = 0;

void draw() {cout << "Graphics的draw()被调用" << endl;};

~Graphics(){ cout << "Graphics类对象被析构" << endl << endl;}

};

class Points : public Graphics{

protected:

int x; int y;

public:

Points() :x(0), y(0){};Points(int x0, int y0) :x(x0), y(y0){};

double area(){return 0;};

virtual void draw(){

cout << "Points的draw()被调用" << endl; cout << "x=" << x << ",y=" << y << endl;

}

virtual ~Points(){ cout << "Points类对象被析构" << endl; }

};

class Circle : public Points

{

private:

int r;

public:

Circle() :Points(), r(0){};

Circle(int x0, int y0, int r0) :Points(x0, y0), r(r0){};

double area(){return 3.14\*r\*r;};

void draw(){

cout << "Circle的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",r=" << r << endl;

cout << "Circle的面积为：" << area() << endl;

}

~Circle(){ cout << "Circle类对象被析构" << endl; }

};

class Ellipse : public Points{

private:

int a; int b;

public:

Ellipse() :Points(), a(0), b(0){};

Ellipse(int x0, int y0, int a0, int b0) :Points(x0, y0), a(a0), b(b0){};

double area(){ return 3.14\*a\*b; };

void draw(){

cout << "Ellipse的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",a=" << a << ",b=" << b << endl;

cout << "Ellipse的面积为：" << area() << endl;

}

~Ellipse(){ cout << "Ellipse类对象被析构" << endl; }

};

int main()

{

Graphics \*ptr;

cout << "Graphics类指针指向派生类：" << endl << endl;

ptr = new Points(1, 1);ptr->draw(); delete ptr;

ptr = new Circle(2, 2, 4); ptr->draw();delete ptr;

ptr = new Ellipse(3, 3, 2, 1); ptr->draw();delete ptr;

Points \*p\_ptr;

cout << "Points类指针指向派生类：" << endl << endl;

p\_ptr = new Circle(4, 4, 8);p\_ptr->draw(); delete p\_ptr;

p\_ptr = new Ellipse(6, 6, 4, 2); p\_ptr->draw();delete p\_ptr;

return 0;

}

程序3

#include <iostream>

#include <string.h>

using namespace std;

class Graphics{

public:

double area(){};

void draw() { cout << "Graphics的draw()被调用" << endl; };

~Graphics(){ cout << "Graphics类对象被析构" << endl << endl; }

};

class Points : public Graphics{

protected:

int x; int y;

public:

Points() :x(0), y(0){};

Points(int x0, int y0) :x(x0), y(y0){};

double area(){return 0;};

void draw(){

cout << "Points的draw()被调用" << endl; cout << "x=" << x << ",y=" << y << endl;

}

~Points(){ cout << "Points类对象被析构" << endl; }

};

class Circle : public Points{

private:

int r;

public:

Circle() :Points(), r(0){};

Circle(int x0, int y0, int r0) :Points(x0, y0), r(r0){};

double area(){ return 3.14\*r\*r;};

void draw(){

cout << "Circle的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",r=" << r << endl;

cout << "Circle的面积为：" << area() << endl;

}

~Circle(){ cout << "Circle类对象被析构" << endl; }

};

class Ellipse : public Points{

private:

int a;int b;

public:

Ellipse() :Points(), a(0), b(0){};

Ellipse(int x0, int y0, int a0, int b0) :Points(x0, y0), a(a0), b(b0){};

double area(){ return 3.14\*a\*b; };

void draw(){

cout << "Ellipse的draw()被调用" << endl;

cout << "x=" << x << ",y=" << y << ",a=" << a << ",b=" << b << endl;

cout << "Ellipse的面积为：" << area() << endl;

}

~Ellipse(){ cout << "Ellipse类对象被析构" << endl; }

};

int main(){

Graphics \*ptr;

cout << "Graphics类指针指向派生类：" << endl << endl;

ptr = new Points(1, 1);ptr->draw(); delete ptr;

ptr = new Circle(2, 2, 4);ptr->draw();delete ptr;

ptr = new Ellipse(3, 3, 2, 1);ptr->draw();delete ptr;

Points \*p\_ptr;

cout << "Points类指针指向派生类：" << endl << endl;

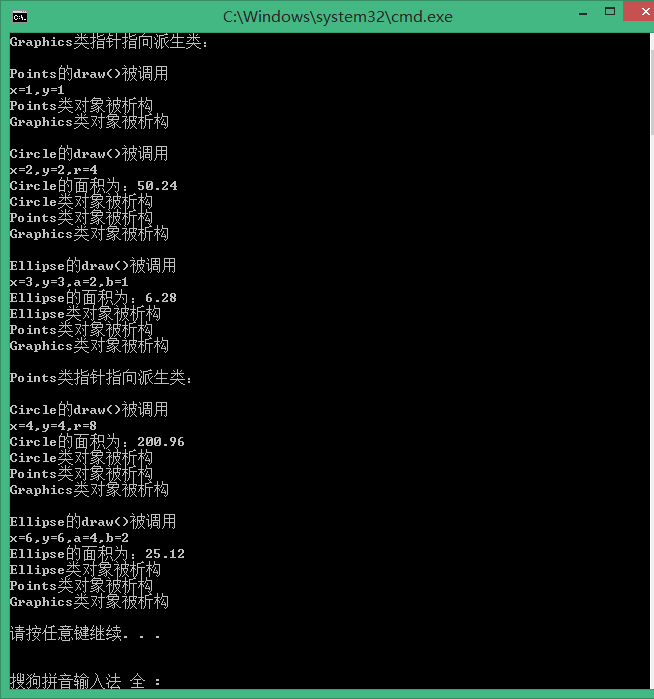
p\_ptr = new Circle(4, 4, 8);p\_ptr->draw();delete p\_ptr;

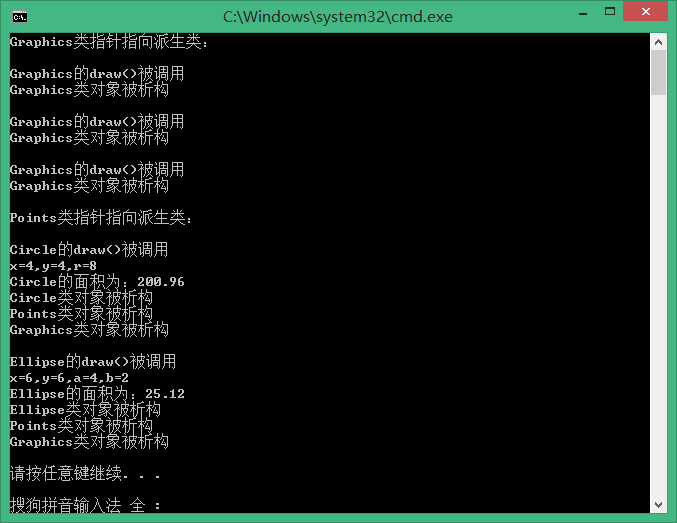
p\_ptr = new Ellipse(6, 6, 4, 2); p\_ptr->draw();delete p\_ptr;

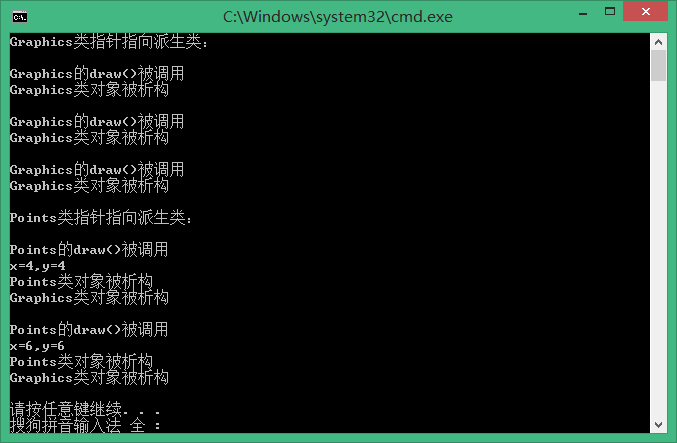
return 0;

}

1. **结果分析与体会：**







**代码分析：**

三个程序的主程序main函数相同，程序1中将Graphic中的函数声明为了虚函数，程序2中将Points类中的函数声明为了虚函数，而程序3中无虚函数。三个程序的继承关系均相同。主函数中分别使用了Graphic基类和Points基类的指针指向了派生类的对象，其中Points类又是Graphic类的子类。

**结果分析：**

程序1的输出结果为Graphic指针指向的派生类对象的函数调用都分别调用了自己类中的相应的函数，Points指针指向的派生类对象的函数调用也都分别调用了自己类中的相应成员函数。析构函数的调用顺序也是先派生类，后基类层层调用。

程序2的输出结果为Graphic指针指向的派生类对象的函数调用都调用了基类Graphic类中本身的函数，析构函数也是如此。而Points类指针指向的派生类对象的函数调用都分别调用了派生类中各自的成员函数，析构函数也是先派生类后基类的顺序层层调用。

程序3的输出结果为Graphic指针和Points指针指向的派生类对象的函数调用都只是调用了指针类型对应的类中的成员函数，而没有调用派生类对象中的成员函数。析构函数也是如此。只是Points指针指向的对象需要先调用Points类的析构函数再调用Graphic的析构函数，因为Points类继承Graphic类。

**结论和体会：**

虚函数的作用是允许在派生类中重新定义与基类同名的函数，并且可以通过基类指针或引用访问基类和派生类中的同名函数。在派生类的基类部分中，派生类的函数取代了基类原来的虚函数，因此在使基类指针指向派生类对象后，调用函数时就调用了派生类的函数。定义一个指向基类对象的指针变量，并使它指向同一类族中需要调用该函数的对象。通过该指针变量调用此函数，此时调用的就是指针变量指向的对象的同名函数。

程序1中Graphic类中的函数为虚函数，而Points类中的函数为普通函数，但是Points指针指向的派生类对象仍然可以访问派生类对象自身的函数，说明Points类中的函数也是虚特性的，Points类之所以具有虚特性，是因为Points类继承了Graphic类，Graphic类具有虚特性，而虚特性是可以被继承的。但虚特性的继承要求函数原型完全相同。