**1.**

#include <iostream>

using namespace std;

#define PI 3.1415926535897932384626

class Point

{

public:

Point(double \_x, double \_y) : x(\_x), y(\_y) { cout << "Point constructor!" << endl; }

Point(const Point& \_pt) : x(\_pt.x), y(\_pt.y) { cout << "Point copy constructor!" << endl; }

virtual double GetArea() const { return 0.0; }

~Point() { cout << "Point destructor!" << endl; }

private:

double x;

double y;

};

class Circle : public Point

{

public:

Circle(double \_x, double \_y, double \_R) : Point(\_x, \_y), R(\_R) { cout << "Circle constructor!" << endl; }

Circle(const Circle& \_cl) : Point(\_cl), R(\_cl.R) { cout << "Circle copy constructor!" << endl; }

virtual double GetArea() const { return PI \* R \* R; }

~Circle() { cout << "Circle destructor!" << endl; }

private:

double R;

};

class Rectangle : public Point

{

public:

Rectangle(double \_x, double \_y, double \_height, double \_width) : Point(\_x, \_y), Height(\_height), Width(\_width) { cout << "Rectangle constructor!" << endl; }

Rectangle(const Rectangle& \_rec) : Point(\_rec), Height(\_rec.Height), Width(\_rec.Width) { cout << "Rectangle copy constructor!" << endl; }

virtual double GetArea() const { return Height \* Width; }

~Rectangle() { cout << "Rectangle destructor!" << endl; }

private:

double Height;

double Width;

};

class Square : public Rectangle

{

public:

Square(double \_x, double \_y, double \_side) : Rectangle(\_x, \_y, \_side, \_side) { cout << "Square constructor" << endl; }

Square(const Square& \_sq) : Rectangle(\_sq) { cout << "Square copy constructor!" << endl; }

virtual double GetArea() const { return Rectangle::GetArea(); }

~Square() { cout << "Square destructor!" << endl; }

};

void f(Point& a)

{

cout << a.GetArea() << endl;

}

int main()

{

Point p1(0, 1);

Circle c1(1, 1, 5);

Rectangle r1(2, 3, 5, 6);

Square s1(9, 4, 2);

f(p1);

f(c1);

f(r1);

f(s1);

Point p2(p1);

Circle c2(c1);

Rectangle r2(r1);

Square s2(s1);

cout << p2.GetArea() << " " << c2.GetArea() << " " << r2.GetArea() << " " << s2.GetArea() << endl;

Point\* ptptr = new Point(0, 5);

Circle\* clptr = new Circle(2, 3, 5);

Rectangle\* rtptr = new Rectangle(8, 7, 1, 2);

Square\* sqptr = new Square(6, 3, 5);

Point\* ptr[4];

ptr[0] = ptptr;

ptr[1] = clptr;

ptr[2] = rtptr;

ptr[3] = sqptr;

for (int i = 0; i < 4; ++i)

cout << ptr[i]->GetArea() << " ";

delete sqptr;

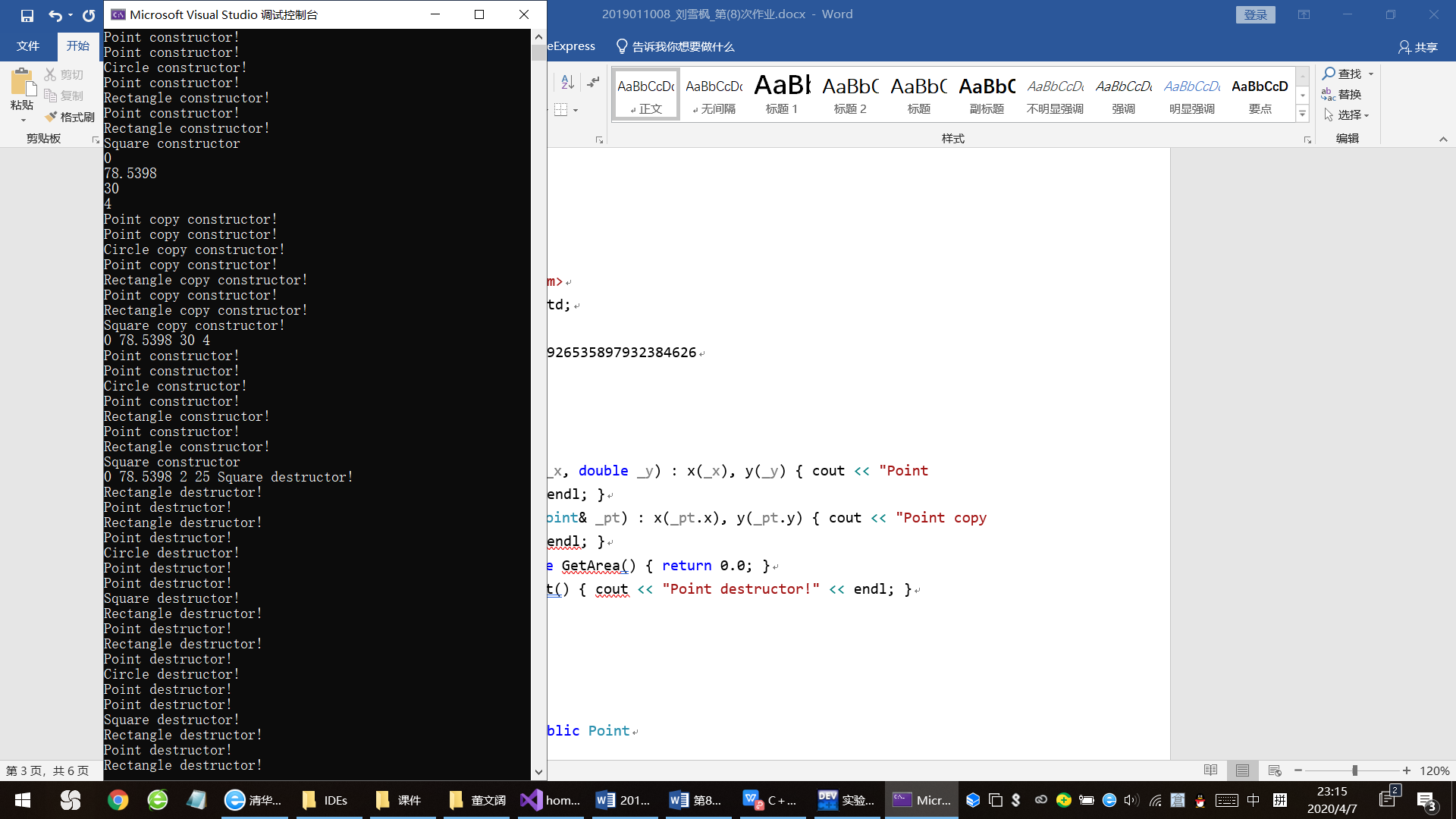
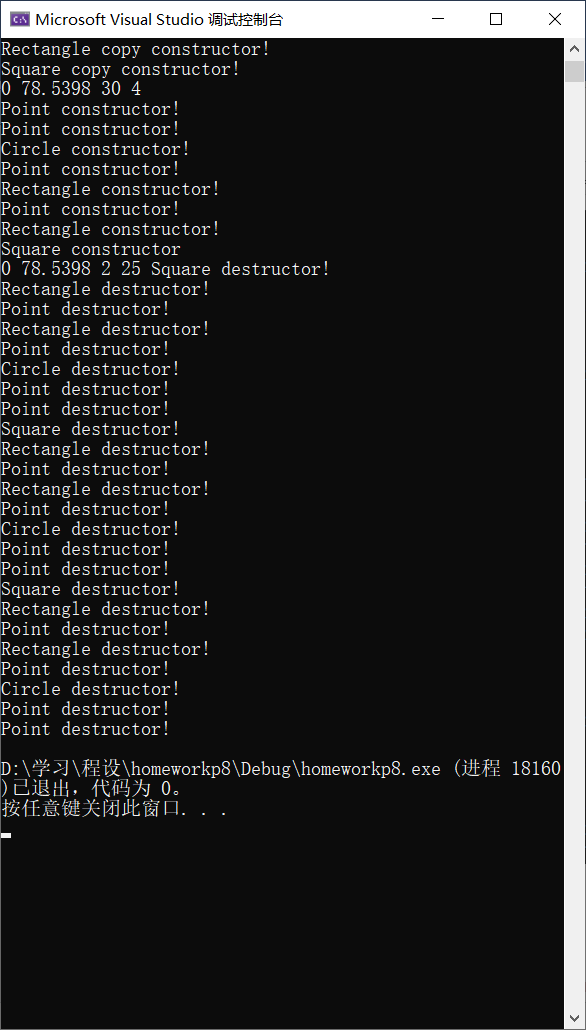
delete rtptr;

delete clptr;

delete ptptr;

return 0;

}

**2.**

#include <iostream>

using namespace std;

#define PI 3.1415926535897932384626

class Point

{

public:

Point(double \_x, double \_y) : x(\_x), y(\_y) { cout << "Point constructor!" << endl; }

Point(const Point& \_pt) : x(\_pt.x), y(\_pt.y) { cout << "Point copy constructor!" << endl; }

virtual double GetArea() const { return 0.0; }

virtual ~Point() { cout << "Point destructor!" << endl; }

private:

double x;

double y;

};

class Circle : public Point

{

public:

Circle(double \_x, double \_y, double \_R) : Point(\_x, \_y), R(\_R) { cout << "Circle constructor!" << endl; }

Circle(const Circle& \_cl) : Point(\_cl), R(\_cl.R) { cout << "Circle copy constructor!" << endl; }

virtual double GetArea() const { return PI \* R \* R; }

virtual ~Circle() { cout << "Circle destructor!" << endl; }

private:

double R;

};

class Rectangle : public Point

{

public:

Rectangle(double \_x, double \_y, double \_height, double \_width) : Point(\_x, \_y), Height(\_height), Width(\_width) { cout << "Rectangle constructor!" << endl; }

Rectangle(const Rectangle& \_rec) : Point(\_rec), Height(\_rec.Height), Width(\_rec.Width) { cout << "Rectangle copy constructor!" << endl; }

virtual double GetArea() const { return Height \* Width; }

virtual ~Rectangle() { cout << "Rectangle destructor!" << endl; }

private:

double Height;

double Width;

};

class Square : public Rectangle

{

public:

Square(double \_x, double \_y, double \_side) : Rectangle(\_x, \_y, \_side, \_side) { cout << "Square constructor" << endl; }

Square(const Square& \_sq) : Rectangle(\_sq) { cout << "Square copy constructor!" << endl; }

virtual double GetArea() const { return Rectangle::GetArea(); }

virtual ~Square() { cout << "Square destructor!" << endl; }

};

void f(Point& a)

{

cout << a.GetArea() << endl;

}

int main()

{

Point p1(0, 1);

Circle c1(1, 1, 5);

Rectangle r1(2, 3, 5, 6);

Square s1(9, 4, 2);

f(p1);

f(c1);

f(r1);

f(s1);

Point p2(p1);

Circle c2(c1);

Rectangle r2(r1);

Square s2(s1);

cout << p2.GetArea() << " " << c2.GetArea() << " " << r2.GetArea() << " " << s2.GetArea() << endl;

Point\* ptr[4];

ptr[0] = new Point(0, 5);

ptr[1] = new Circle(2, 3, 5);

ptr[2] = new Rectangle(8, 7, 1, 2);

ptr[3] = new Square(6, 3, 5);

for (int i = 0; i < 4; ++i)

cout << "ptr[" << i << "]: " << ptr[i]->GetArea() << endl;

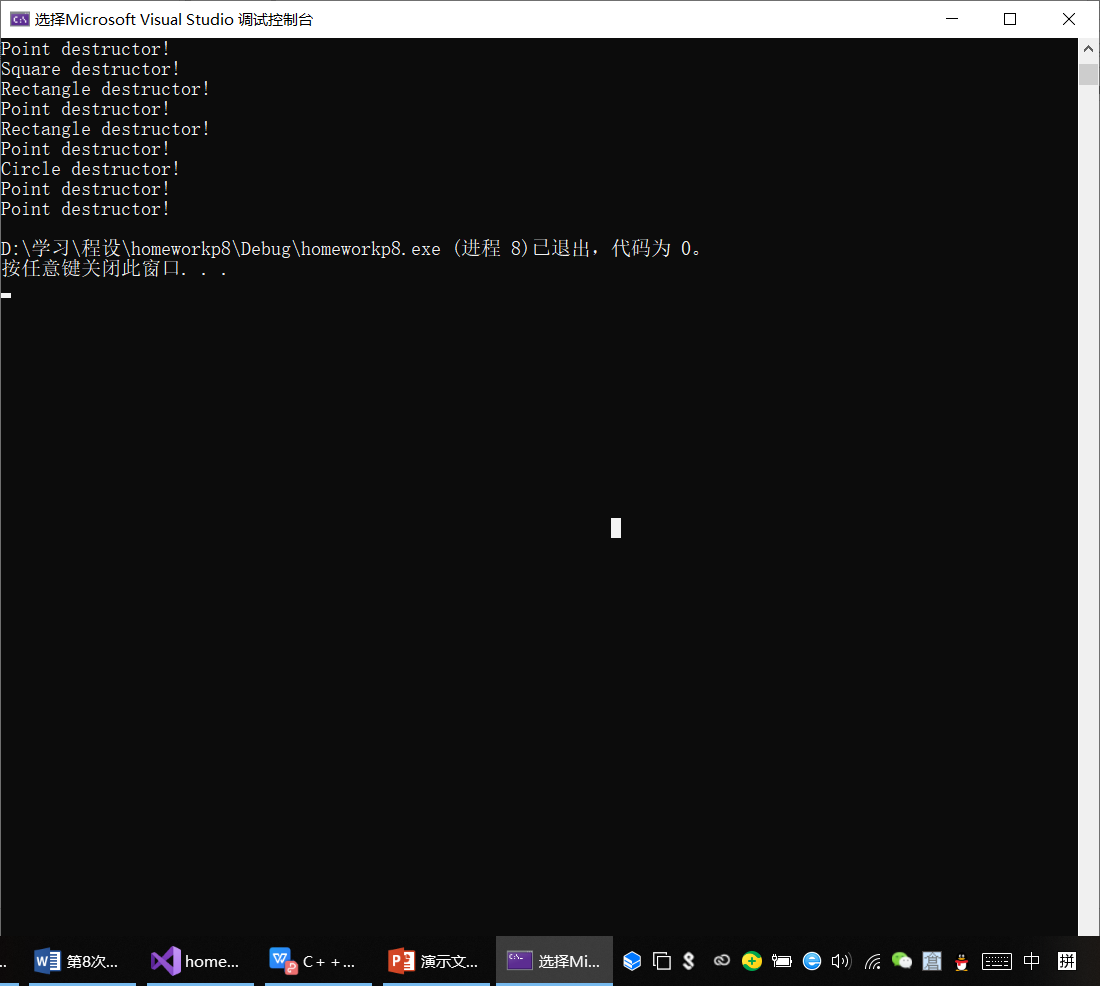
for (int i = 3; i >= 0; --i)

delete ptr[i];

return 0;

}





**3\*.**

#include <iostream>

using namespace std;

#define PI 3.1415926535897932384626

inline double Abs(double x)

{

return x < 0.0 ? -x : x;

}

class Point

{

public:

Point(double \_x, double \_y) : x(\_x), y(\_y) {}

Point(const Point& \_pt) : x(\_pt.x), y(\_pt.y) {}

virtual double GetArea() const { return 0; }

virtual ~Point() {}

private:

double x;

double y;

};

class Circle : virtual public Point

{

public:

Circle(double \_x, double \_y, double \_R) : Point(\_x, \_y), R(\_R) {}

Circle(const Circle& \_cl) : Point(\_cl), R(\_cl.R) {}

virtual double GetArea() const { return PI \* R \* R; }

virtual ~Circle() {}

private:

double R;

};

class Square : virtual public Point

{

public:

Square(double \_x, double \_y, double \_L) : Point(\_x, \_y), L(\_L) {}

Square(const Square& \_sq) : Point(\_sq), L(\_sq.L) {}

virtual double GetArea() const { return L \* L; }

virtual ~Square() {}

private:

double L;

};

class Margin : public Circle, public Square

{

public:

Margin(double \_x, double \_y, double \_R, double \_L) : Point(\_x, \_y), Circle(\_x, \_y, \_R), Square(\_x, \_y, \_L) {}

Margin(const Margin& \_mg) : Point(\_mg), Circle(\_mg), Square(\_mg) {}

virtual double GetArea() const { return Abs(Circle::GetArea() - Square::GetArea()); }

virtual ~Margin() {}

};

int main()

{

Margin m1(0, 0, 5, 1);

Margin m2(1, 2, 1, 20);

cout << "m1: " << m1.GetArea() << endl << "m2: " << m2.GetArea() << endl;

return 0;

}

