**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

по дисциплине «Объектно-ориентированное программирование»

на тему: «Основы синтаксиса объектно-ориентированного

языка программирования»

Выполнил: студент гр. ИТП-21

Воробьев В. А.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2022

**Цель:** необходимо разработать класс, представляющий собой плоскую фигуру, согласно варианта.

**Ход работы**

1. При именовании класса, полей и методов руководствоваться соглашением о наименовании кода.

2. Фигуру следует задавать координатами вершин.

3. Реализовать метод проверки возможности существования данной фигуры

4. Реализовать методы вычисление длин сторон, площади и периметра

5. Реализовать методы, проверяющие принадлежность точки, заданной своими координатами на плоскости, фигуре, её границе.

6. Класс должен быть размещён в библиотеке классов.

Согласно варианта, рассматриваемая фигура ­–– Ромб. Координаты задаются по часовой стрелке. Необходимо реализовать методы, которые:

1. Проверяют, существует ли заданная фигура; в нашем случае нам нужно проверить: попарно ли параллельны стороны ромба, если да – существует.
2. Площадь, которая считается по формуле *S* = *d1* \* *d2*, где *d1, d2*– диагонали.
3. Длины сторон, периметр.
4. Метод, проверяющий, лежит ли заданная точка внутри фигуры или нет.

Чтобы проверить параллельность прямых, нам нужно для начала составить их уравнения и оттуда взять угловые коэффициенты и попарно сравнить: если они равны, то параллельны, иначе – нет. Коэффициенты вычисляются по формуле, где *M1*(*x1*, *y1*), *M2*(*x2*, *y2*) – координаты смежных вершин ромба:

Чтобы посчитать длины сторон, необходимо воспользоваться следующей формулой, где *M1*(*x1*, *y1*), *M2*(*x2*, *y2*) – координаты смежных вершин ромба:

Чтобы проверить, лежит ли точка внутри фигуры, нам необходимо составить уравнение стороны ромба по двум точкам и подставить координаты в формулу, если в итоге получится число больше 0, то точка лежит выше, меньше 0 – ниже, равно – лежит на прямой. В нашем случае точка будет лежать внутри, если она ниже 1-ой, 2-ой прямой и выше 3-ей и 4-ей. Формула выглядит следующим образом, где *M1*(*x1*, *y1*), *M2*(*x2*, *y2*) – координаты смежных вершин ромба, *x*и *y*– координаты точки:

Допустим, наш ромб имеет координаты вершин (0; 0), (5; 5), (10; 5), (5; 0), то для него тогда длина стороны будет равна 7, периметр 28, площадь 27.8. Если ввести эти данные в программу, то результат будет следующим:

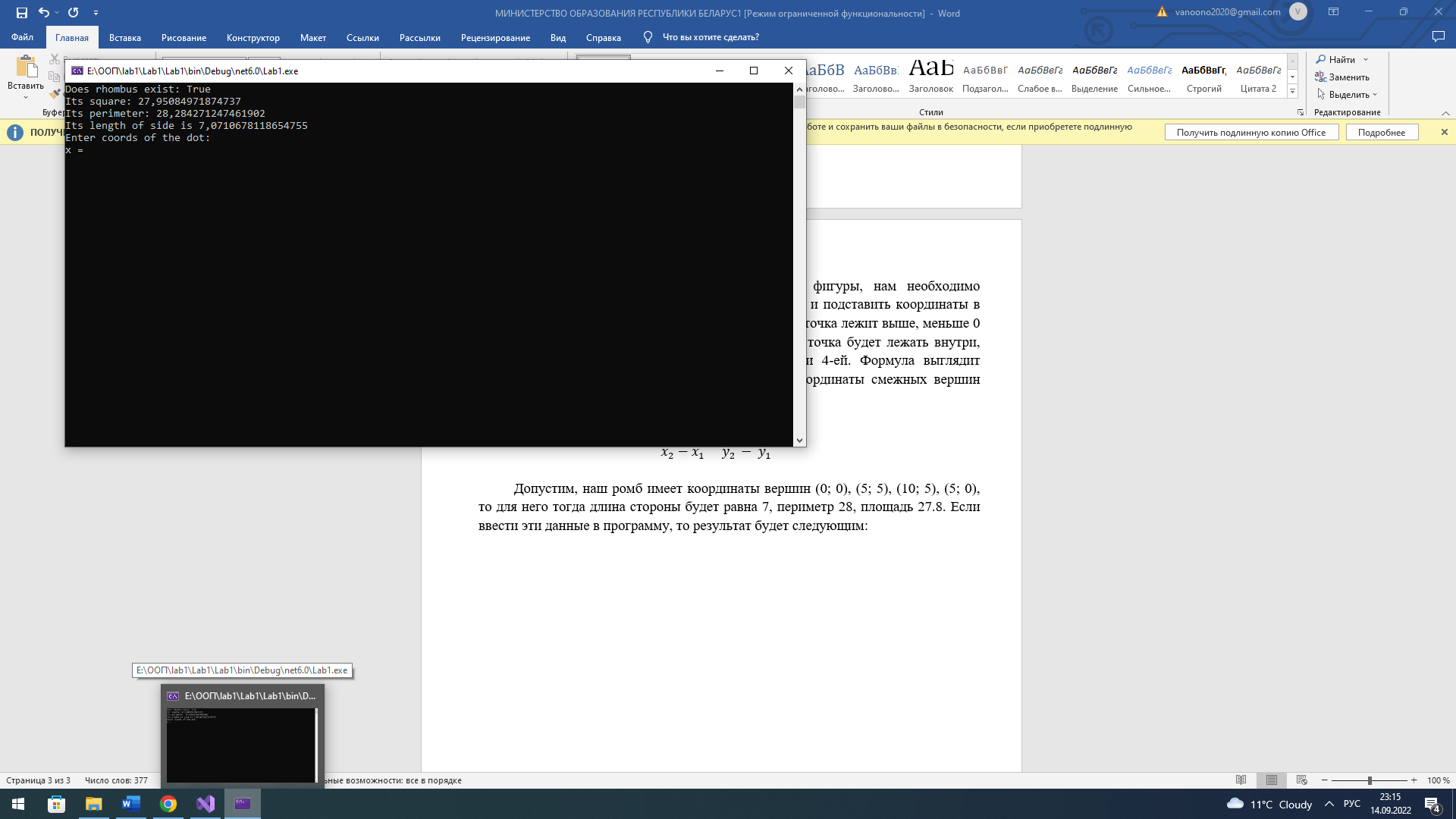


Рисунок 1 – Результат ввода

данных в программу

Если мы введем точку с координатами (4; 2), то она будет лежать внутри, а если, например, (10; 2) – снаружи:

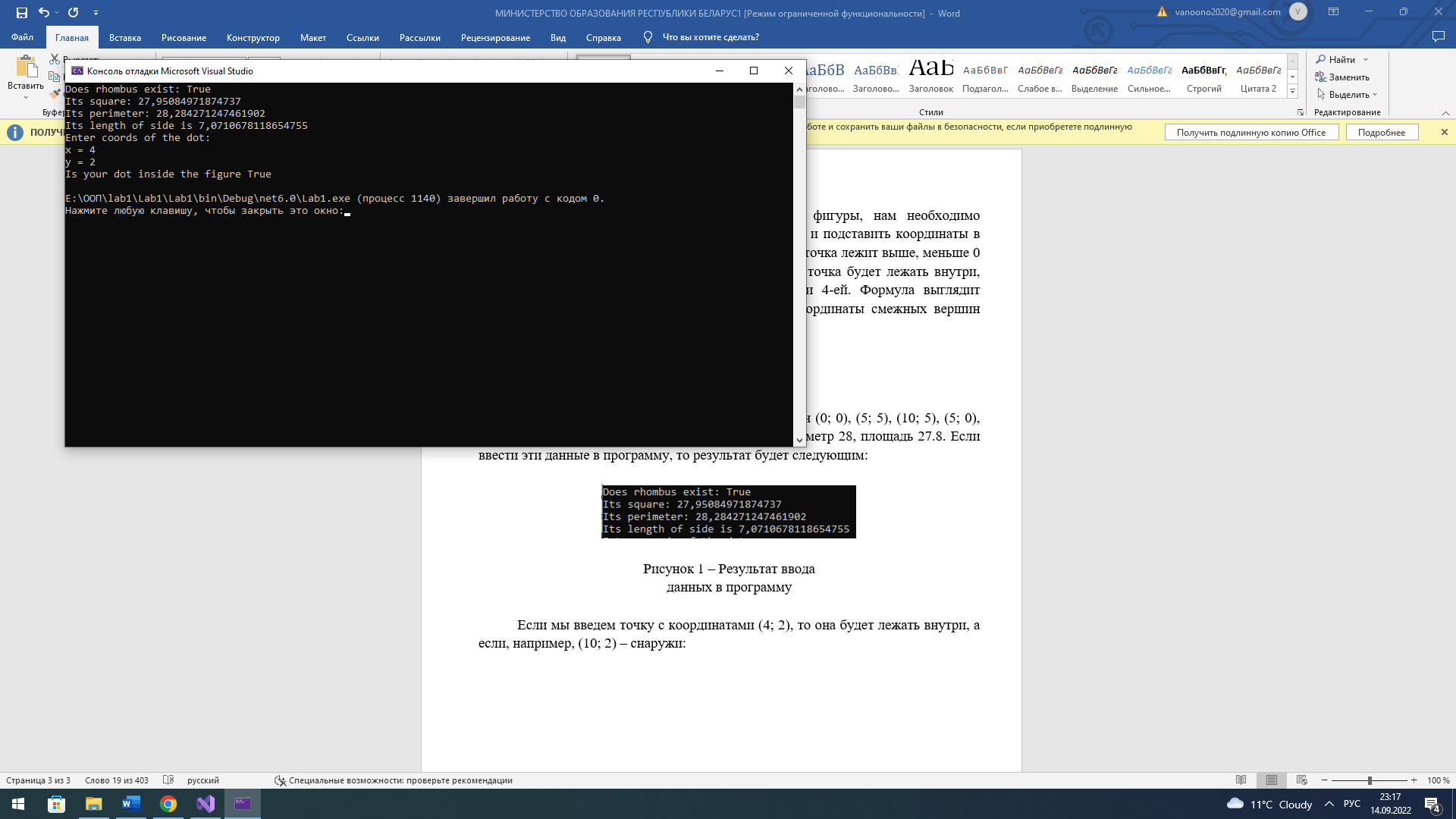


Рисунок 2 – Пример точки, лежащей

внутри фигуры

Ниже же представлена точка, что будет снаружи:

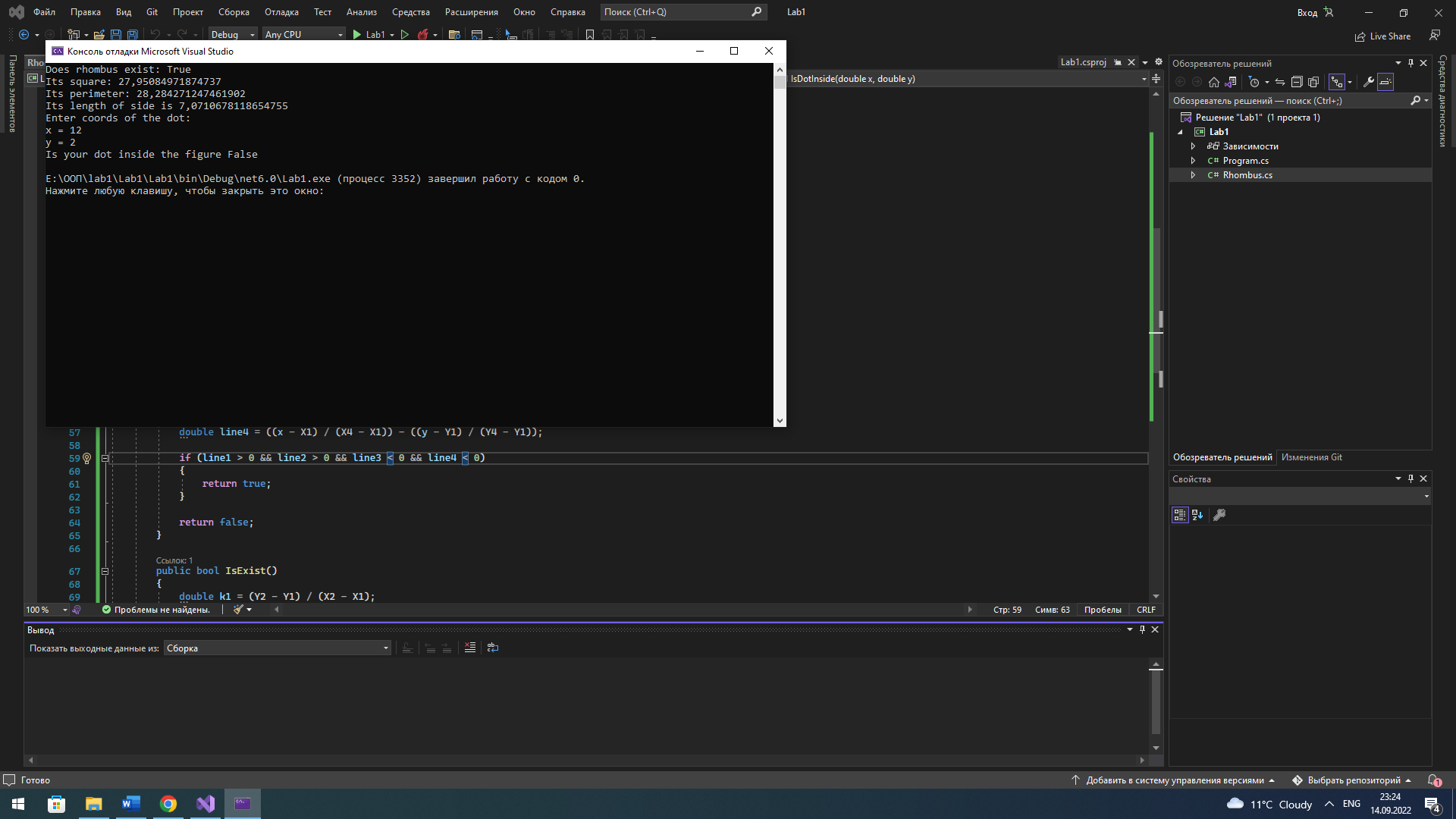


Рисунок 3 – Пример точки, лежащей

снаружи фигуры

Ниже точка, что лежит на одной из сторон:

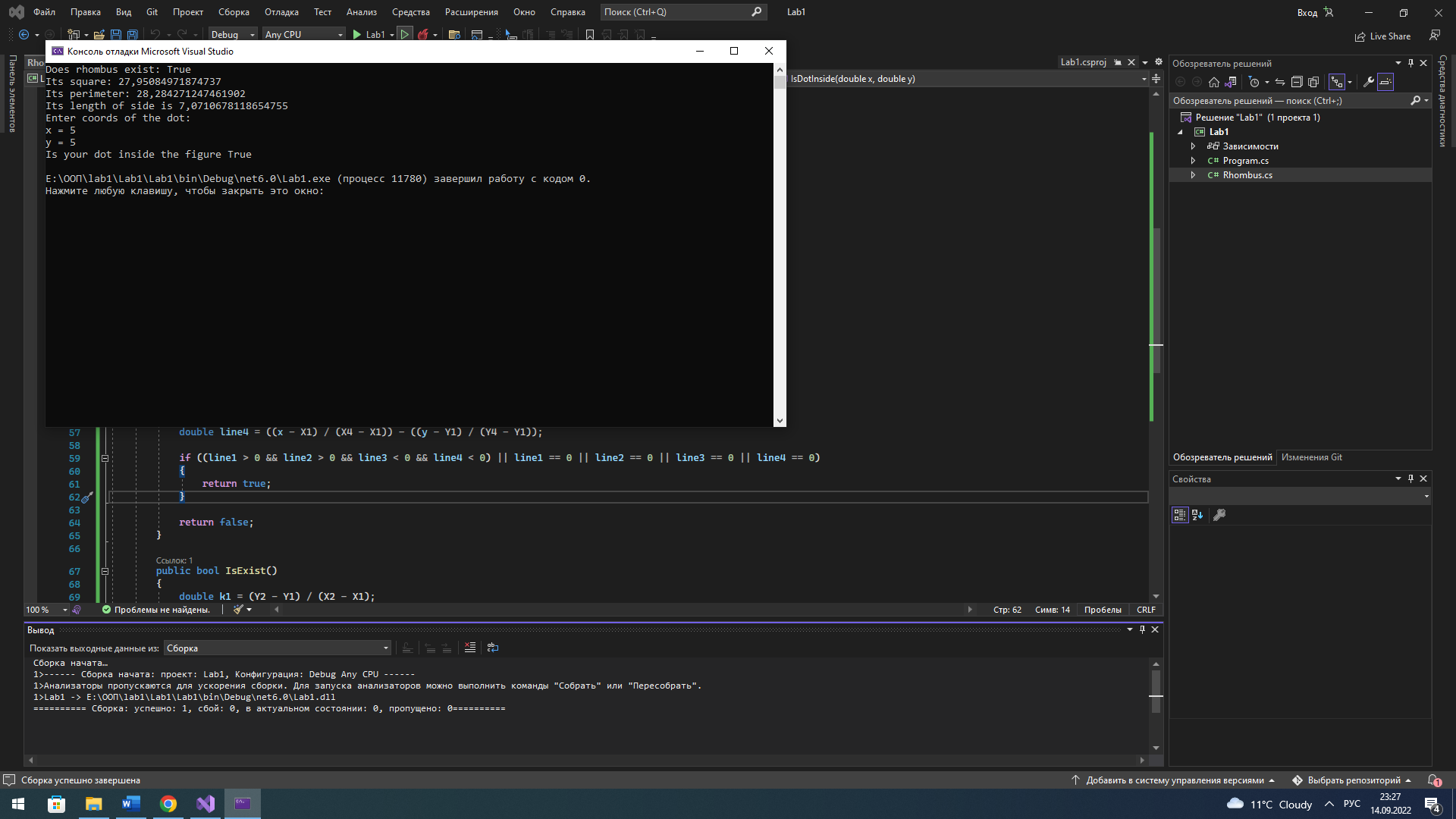


Рисунок 4 – Пример точки, лежащей

на одной из сторон

В приложении А представлен код программы.

**Вывод:** в ходе работы был написан класс, реализующий работу с геометрической фигурой «Ромб». Были изучены основы синтаксиса языка программирования C# и основные принципы объектно-ориентированного программирования.

**ПРИЛОЖЕНИЕ А**

**КОД КЛАССА РОМБ**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

internal class Rhombus

{

public double X1, Y1, X2, Y2, X3, Y3, X4, Y4;

public Rhombus(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

X1 = x1;

Y1 = y1;

X2 = x2;

Y2 = y2;

X3 = x3;

Y3 = y3;

X4 = x4;

Y4 = y4;

}

public double CalcSide()

{

double side = Math.Sqrt(Math.Pow(X2 - X1, 2) + Math.Pow(Y2 - Y1, 2));

return side;

}

public double CalcSquare()

{

double d1 = Math.Sqrt(Math.Pow(X3 - X1, 2) + Math.Pow(Y3 - Y1, 2));

double d2 = Math.Sqrt(Math.Pow(X4 - X2, 2) + Math.Pow(Y4 - Y2, 2));

double S = d1 \* d2 \* 0.5;

return S;

}

public double CalcPerimeter()

{

double side = CalcSide();

double P = side \* 4;

return P;

}

public bool IsDotInside(double x, double y)

{

double line1 = ((x - X1) / (X2 - X1)) - ((y - Y1) / (Y2 - Y1));

double line2 = ((x - X3) / (X2 - X3)) - ((y - Y3) / (Y2 - Y3));

double line3 = ((x - X4) / (X3 - X4)) - ((y - Y3) / (Y4 - Y3));

double line4 = ((x - X1) / (X4 - X1)) - ((y - Y1) / (Y4 - Y1));

if ((line1 > 0 && line2 > 0 && line3 < 0 && line4 < 0) || line1 == 0 || line2 == 0 || line3 == 0 || line4 == 0)

{

return true;

}

return false;

}

public bool IsExist()

{

double k1 = (Y2 - Y1) / (X2 - X1);

double k2 = (Y3 - Y2) / (X3 - X2);

double k3 = (Y4 - Y3) / (X4 - X3);

double k4 = (Y4 - Y1) / (X4 - Y1);

if ((k1 != k3) || (k2 != k4))

{

return false;

}

return true;

}

}

}