

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 423 |

Отчёт по контрольной работе №1

Вариант № 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 423 |  |  |  | Ефремов Иван Андреевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Макарук Роман Валерьевич |
|  |  |  |  | Федин Алексей Константинович |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc134647499)

[2 Исходные данные 3](#_Toc134647500)

[3 Особые ситуации 3](#_Toc134647501)

[4 Математические методы и алгоритмы решения задач 4](#_Toc134647502)

[5 Форматы представления данных 4](#_Toc134647503)

[6 Структура программы 5](#_Toc134647504)

[7 Блок-схема алгоритма решения задачи 9](#_Toc134647505)

[8 Описание хода выполнения 13](#_Toc134647506)

[9 Результаты работы программы 14](#_Toc134647507)

[10 Выводы по заданию 16](#_Toc134647508)

[11 Исходный код полученного программного решения 16](#_Toc134647509)

## 1 Постановка задачи

Для заданных отрезков на плоскости определить, пересекаются ли они. Найти координаты точки пересечения.

## 2 Исходные данные

Исходные данные состоят из координат точек, определяющих концы отрезков. Данные могут быть введены пользователем с консоли, из файла или сгенерированы случайным образом.

## 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

-если при запросе числа пользователь ввёл другие символы, то программа укажет на необходимость ввода рациональных чисел и попросит повторить ввод;

-если при запросе выбрать пункт меню пользователь введет несуществующий вариант, то программа не отреагирует и пользователю придется вновь выбрать пункт меню;

-если пользователь попытается открыть несуществующий файл для чтения, программа сообщит, что такого файла не существует, и попросит пользователя указать файл для чтения еще раз;

-если пользователь пытается отрыть файл с запрещенным названием для ввода или записи, то программа сообщит о недопустимости такого названия файла;

-если пользователь захочет сохранить данные в файл, в котором уже содержится какая-то информация, то программа спросит, хочет ли пользователь перезаписать данные в этом файле;

-если файле для считывания будет пуст или в нем будет некорректное количество координат, то программа сообщит об ошибке и попросит заново выбрать файл.

## 4 Математические методы и алгоритмы решения задач

Согласно постановке задачи, для получения необходимых результатов будут использоваться операторы “>”, “≥”, “<”, “≤”, “=”, “+”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 Форматы представления данных Таблица 1 - Основные переменные, используемые в программе   |  |  |  | | --- | --- | --- | | Имя | Тип | Описание | | first | Segment | Объект класса Segment, хранящий информацию о концах первого отрезка | | second | Segment | Объект класса Segment, хранящий информацию о концах второго отрезка | | res | Segment | Объект класса Segment, хранящий информацию о точке/области пересечения |   Таблица 2 - Класс Segment   |  |  |  |  | | --- | --- | --- | --- | | Имя | Описание | Модификатор | Тип | | start | Точка начала отрезка | public | Point | | end | Точка конца отрезка | public | Point |    6 Структура программы Таблица 3 - Модули программы   |  |  | | --- | --- | | Название модуля | Описание | | Program | Зацикленное меню и основное тело программы | | Information | Вывод сообщений о работе программы в консоль | | Segment | Файл, содержащий классы Point и Segment | | DataInputAndOutput | Ввод данных с файла/консоли/случайными значениями, вывод в файл | | SegmentIntersection | Алгоритм поиска точки/области пересечения отрезков |   Таблица 4 - Содержимое модуля Segment   |  |  |  | | --- | --- | --- | | Тип | Имя | Описание | | class | Point | Класс, описывающий точку на плоскости | | class | Segment | Класс, описывающий отрезок на плоскости | |  |

Таблица 5 – Основные функции модуля Information

|  |  |
| --- | --- |
| Имя | Описание |
| Greeting | Вывод приветственного сообщения |
| Options | Вывод опций программы |
| AskToSave | Вывод сообщения, хочет ли пользователь сохранить данные в файл. |
| FileInputInfo | Вывод сообщения с требованиями к расположению информации в файле |
| AskToOverwrite | Вывод сообщения, хочет ли пользователь перезаписать данные в файл. |
| SaveSuccess | Вывод сообщения об успешном сохранении данных в файл |

Таблица 6 – Функции модуля DataInputAndOutput

|  |  |
| --- | --- |
| Имя | Описание |
| ReadDouble | Считывание рационального числа из консоли |
| ReadPath | Считывание пути к файлу |
| SplitString | Разделение строки по пробелам при ввода с файла |
| SaveData | Сохранение данных в файл |
| ConsoleInput | Ввод данных с консоли |
| RandomInput | Генерация случайных данных |
| FileInput | Ввода данных с файла |

Таблица 7 – Функции модуля SegmentIntersection

|  |  |
| --- | --- |
| Имя | Описание |
| OneVerticalSegment | Алгоритм поиска точки/области пересечения, при одном вертикальном отрезке |
| Intersection | Алгоритм поиска точки/области пересечения |

## 7 Блок-схема алгоритма решения задачи

На рисунке 1 представлена блок-схема алгоритма для шифра Цезаря.



Рисунок 1 – Блок-схема алгоритма

## 8 Описание хода выполнения

Во время работы, программа предлагает пользователю три варианта ввода данных (из консоли, из файла и генерация случайных значений). Если пользователь выбрал любой вариант ввода, кроме ввода из файла. То после ввода/генерации значений программа предложит пользователю сохранить эти значения в файл. После этого программа рассчитает точку/область пересечения и выведет сообщение. Если отрезки не пересекаются, то программа выведет «Segments do not intersect». Если отрезки пересекаются в одной точке, то программа выведет «Segments intersect at {координаты точки}». Если отрезки пересекаются на области, то программа выведет «Segments intersect within {координаты начала области}: {координаты конца области}». После этого программа предложит пользователю сохранить результат работы в файл.

## 9 Результаты работы программы

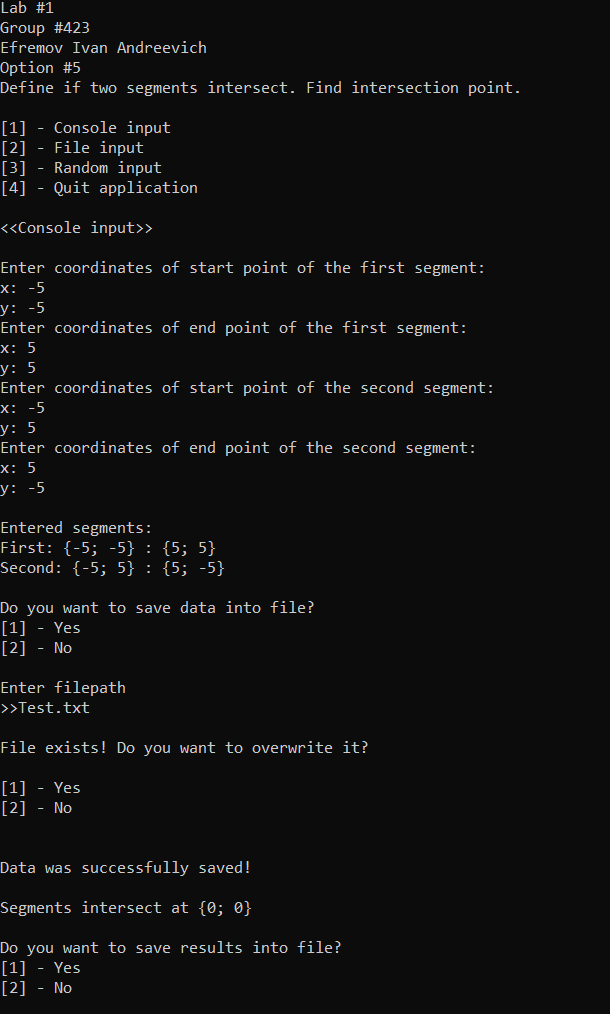


Рисунок 2 – Экранная копия результата работы разработанной программы

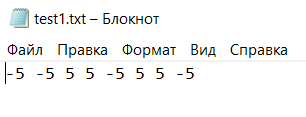


Рисунок 3 – Данные, сохраненные в файл

## 10 Выводы по заданию

В ходе выполнения задания были изучены основы работы с языком С# и реализацией классов, а также были получены навыки работы с многофайловыми проектами.

## 11 Исходный код полученного программного решения

Файл Program.cs:

using Microsoft.VisualBasic;

namespace Lab1

{

internal class Program

{

static void Main()

{

Segment first = new();

Segment second = new();

Information.Greeting();

Information.Options();

while (true)

{

switch (Console.ReadKey(true).Key)

{

case ConsoleKey.D1:

DataInputAndOutput.ConsoleInput(ref first, ref second);

break;

case ConsoleKey.D2:

DataInputAndOutput.FileInput(ref first, ref second);

break;

case ConsoleKey.D3:

DataInputAndOutput.RandomInput(ref first, ref second);

break;

case ConsoleKey.D4:

Console.WriteLine($"{Environment.NewLine}Programm finished it's work!");

Environment.Exit(0);

break;

default:

continue;

}

Segment res = SegmentIntersection.Intersection(first, second);

if (double.IsNaN(res.start.x))

Console.WriteLine($"Segments do not intersect :( {Environment.NewLine}");

else if (double.IsNaN(res.end.x))

Console.WriteLine($"Segments intersect at {{{res.start.x}; " +

$"{res.start.y}}}{Environment.NewLine}");

else

Console.WriteLine($"Segments intersect within " +

$"{{{res.start.x}; {res.start.y}}} : " +

$"{{{res.end.x}; {res.end.y}}}{Environment.NewLine}");

Information.AskToSave("results");

if (Console.ReadKey(true).Key == ConsoleKey.D1) DataInputAndOutput.SaveData([res]);

Information.Options();

}

}

}

}

Файл DataInputAndOutput.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

static class DataInputAndOutput

{

private static double ReadDouble(string message)

{

bool isCorrectInput = false;

double value = 0;

while (!isCorrectInput)

{

Console.Write(message);

string? temp = Console.ReadLine();

if (!double.TryParse(temp, out value))

Console.WriteLine("Invalid input! Numbers only allowed!");

else isCorrectInput = true;

}

return value;

}

private static string ReadPath()

{

string? path = "";

while (string.IsNullOrWhiteSpace(path))

{

Console.Write($"Enter filepath{Environment.NewLine}>>");

path = Console.ReadLine();

if (string.IsNullOrWhiteSpace(path))

Console.WriteLine($"{Environment.NewLine}Path must not be empty!{Environment.NewLine}");

}

return path;

}

private static double[] SplitString(string input)

{

string[] separated = input.Split(' ', StringSplitOptions.RemoveEmptyEntries);

if(separated.Length != 8) return new double[2];

double[] coords = new double[8];

for(int i = 0; i < 8; i++)

if (!double.TryParse(separated[i], out coords[i])) return [i + 1];

return coords;

}

public static void SaveData(Segment[] segments)

{

bool isCorrect = false;

while (!isCorrect)

{

string path = ReadPath();

if (File.Exists(path))

{

Console.WriteLine($"{Environment.NewLine}File exists! Do you want to overwrite it?{Environment.NewLine}");

Information.AskToOverwriteFile();

if(Console.ReadKey(true).Key == ConsoleKey.D2) continue;

}

else

{

try

{

FileStream fs = File.Open(path, FileMode.OpenOrCreate);

fs.Close();

}

catch(IOException)

{

Console.WriteLine($"{Environment.NewLine}Path contains forbidden values!{Environment.NewLine}");

continue;

}

}

if(segments.Length == 1)

{

if (double.IsNaN(segments[0].start.x))

File.WriteAllText(path, "Segments do not intersect");

else if (double.IsNaN(segments[0].end.x))

File.WriteAllText(path, $"Segments intersect at {{{segments[0].start.x}; " +

$"{segments[0].start.y}}}");

else

File.WriteAllText(path, $"Segmets intersect within " +

$"{{{segments[0].start.x}; {segments[0].start.y}}} : " +

$"{{{segments[0].end.x}; {segments[0].end.y}}}");

Information.SaveSuccess("Results");

}

else

{

string data = $"{segments[0].start.x} {segments[0].start.y} " +

$"{segments[0].end.x} {segments[0].end.y} " +

$"{segments[1].start.x} {segments[1].start.y} " +

$"{segments[1].end.x} {segments[1].end.y}";

File.WriteAllText(path, data);

Information.SaveSuccess("Data");

}

isCorrect = true;

}

}

public static void ConsoleInput(ref Segment first, ref Segment second)

{

Console.WriteLine($"<<Console input>>{Environment.NewLine}");

Console.WriteLine("Enter coordinates of start point of the first segment:");

double x1 = ReadDouble("x: ");

double y1 = ReadDouble("y: ");

Console.WriteLine("Enter coordinates of end point of the first segment:");

double x2 = ReadDouble("x: ");

double y2 = ReadDouble("y: ");

Console.WriteLine("Enter coordinates of start point of the second segment:");

double x3 = ReadDouble("x: ");

double y3 = ReadDouble("y: ");

Console.WriteLine("Enter coordinates of end point of the second segment:");

double x4 = ReadDouble("x: ");

double y4 = ReadDouble("y: ");

Console.WriteLine();

first = new Segment(x1, y1, x2, y2);

second = new Segment(x3, y3, x4, y4);

Console.WriteLine(

$"Entered segments:{Environment.NewLine}" +

$"First: {{{first.start.x}; {first.start.y}}} : {{{first.end.x}; {first.end.y}}}{Environment.NewLine}" +

$"Second: {{{second.start.x}; {second.start.y}}} : {{{second.end.x}; {second.end.y}}}{Environment.NewLine}"

);

Information.AskToSave("data");

if(Console.ReadKey(true).Key == ConsoleKey.D1) SaveData([first, second]);

}

public static void RandomInput(ref Segment first, ref Segment second)

{

Console.WriteLine($"<<Random input>>{Environment.NewLine}");

int randomBorder = 999;

double[] vals = new double[8];

var random = new Random();

for(int i = 0; i < 8; i++)

vals[i] = random.Next(-randomBorder, randomBorder);

first = new Segment(vals[0], vals[1], vals[2], vals[3]);

second = new Segment(vals[4], vals[5], vals[6], vals[7]);

Console.WriteLine(

$"Entered segments:{Environment.NewLine}" +

$"First: {{{first.start.x}; {first.start.y}}} : {{{first.end.x}; {first.end.y}}}{Environment.NewLine}" +

$"Second: {{{second.start.x}; {second.start.y}}} : {{{second.end.x}; {second.end.y}}}{Environment.NewLine}"

);

Information.AskToSave("data");

if (Console.ReadKey(true).Key == ConsoleKey.D1) SaveData([first, second]);

}

public static void FileInput(ref Segment first, ref Segment second)

{

Console.WriteLine($"<<File input>>{Environment.NewLine}");

Information.FileInputInfo();

bool isCorrect = false;

while(!isCorrect)

{

string path = ReadPath();

if (path.Intersect(Path.GetInvalidFileNameChars()).Any())

{

Console.WriteLine($"{Environment.NewLine}Path contains forbidden values!{Environment.NewLine}");

continue;

}

if (!File.Exists(path))

{

Console.WriteLine($"{Environment.NewLine}There is no such file!{Environment.NewLine}");

continue;

}

string? input = File.ReadAllText(path);

if (string.IsNullOrWhiteSpace(input))

{

Console.WriteLine($"File is empty!{Environment.NewLine}");

continue;

}

double[] coords = SplitString(input);

if(coords.Length == 2)

{

Console.WriteLine($"Invalid amount of values in the file!{Environment.NewLine}");

continue;

}

if(coords.Length == 1)

{

Console.WriteLine($"Invalid value at position {coords[0]}!{Environment.NewLine}");

continue;

}

first = new Segment(coords[0], coords[1], coords[2], coords[3]);

second = new Segment(coords[4], coords[5], coords[6], coords[7]);

isCorrect = true;

}

Console.WriteLine(

$"{Environment.NewLine}Entered segments:{Environment.NewLine}" +

$"First: {{{first.start.x}; {first.start.y}}} : {{{first.end.x}; {first.end.y}}}{Environment.NewLine}" +

$"Second: {{{second.start.x}; {second.start.y}}} : {{{second.end.x}; {second.end.y}}}{Environment.NewLine}"

);

}

}

}

Файл Information:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

static class Information

{

public static void Greeting()

{

Console.WriteLine("Lab #1");

Console.WriteLine("Group #423");

Console.WriteLine("Efremov Ivan Andreevich");

Console.WriteLine("Option #5");

Console.WriteLine($"Define if two segments intersect. Find intersection point.{Environment.NewLine}");

}

public static void Options()

{

Console.WriteLine($"[1] - Console input");

Console.WriteLine($"[2] - File input");

Console.WriteLine($"[3] - Random input");

Console.WriteLine($"[4] - Quit application");

Console.WriteLine();

}

public static void AskToSave(string message)

{

Console.WriteLine($"Do you want to save {message} into file?");

Console.WriteLine("[1] - Yes");

Console.WriteLine("[2] - No");

Console.WriteLine();

}

public static void FileInputInfo()

{

Console.WriteLine("Data in file must be located in one line. Coordinates must be separated with spaces!");

Console.WriteLine($"Example:{Environment.NewLine}x1 y1 x2 y2 x3 y3 x4 y4{Environment.NewLine}");

}

public static void AskToOverwriteFile()

{

Console.WriteLine("[1] - Yes");

Console.WriteLine("[2] - No");

Console.WriteLine();

}

public static void SaveSuccess(string message)

{

Console.WriteLine();

Console.WriteLine($"{message} was successfully saved!");

Console.WriteLine();

}

}

}

Файл Segment.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

class Point(double x, double y)

{

public double x = x;

public double y = y;

}

class Segment

{

public Point start;

public Point end;

public Segment()

{

start = new Point(0, 0);

end = new Point(0, 0);

}

public Segment(double x1, double y1, double x2, double y2)

{

start = new Point(x1, y1);

end = new Point(x2, y2);

if (x1 > x2) (start, end) = (end, start);

}

}

}

Файл SegmentIntersection.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

static class SegmentIntersection

{

//Определение точки пересечения двух отрезков, один из которых вертикальный

private static Segment OneVerticalSegment(Segment noIntersection, Segment vertical, Segment common)

{

double k = (common.start.y - common.end.y) / (common.start.x - common.end.x);

double b = common.start.y - k \* common.start.x;

double y = k \* vertical.start.x + b;

if (y <= Math.Max(vertical.start.y, vertical.end.y) && y >= Math.Min(vertical.start.y, vertical.end.y))

return new Segment(Math.Round(vertical.start.x, 2), Math.Round(y, 2), double.NaN, double.NaN);

return noIntersection;

}

//Определение точки/промежутка пересечения двух отрезков

public static Segment Intersection(Segment first, Segment second)

{

const double precision = 0.00001;

Segment noIntersection = new(double.NaN, double.NaN, double.NaN, double.NaN);

//Слишком далеко по оси абсцисс

if (first.end.x < second.start.x) return noIntersection;

//Оба отрезка вертикальные

if ((first.end.x - first.start.x < precision) && (second.start.x - second.end.x < precision))

{

//Оба отрезка лежат на одной прямой

if (first.start.x - second.start.x < precision)

{

//Отрезки накладываются

if (!((Math.Max(first.start.y, first.end.y) < Math.Min(second.start.y, second.end.y)) ||

(Math.Min(first.start.y, first.end.y) > Math.Max(second.start.y, second.end.y))))

{

return new Segment(Math.Round(first.start.x, 2),

Math.Round(Math.Max(Math.Min(first.start.y, first.end.y), Math.Min(second.start.y, second.end.y)), 2),

Math.Round(second.start.x, 2),

Math.Round(Math.Min(Math.Max(first.start.y, first.end.y), Math.Max(second.start.y, second.end.y)), 2));

}

return noIntersection;

}

}

//Если только первый отрезок вертикальный

if (first.end.x - first.start.x < precision)

return OneVerticalSegment(noIntersection, first, second);

//Если только второй отрезок вертикальный

if (second.end.x - second.start.x < precision)

return OneVerticalSegment(noIntersection, second, first);

//Общий случай

double k1 = (first.start.y - first.end.y) / (first.start.x - first.end.x);

double k2 = (second.start.y - second.end.y) / (second.start.x - second.end.x);

double b1 = first.start.y - k1 \* first.start.x;

double b2 = second.start.y - k2 \* second.start.x;

//Если угловые коэффициенты не равны

if(k1 - k2 != 0)

{

double x = (b2 - b1) / (k1 - k2);

double y = k1 \* x + b1;

return new Segment(Math.Round(x, 2), Math.Round(y, 2), double.NaN, double.NaN);

}

//Если отрезки накладываются друг на друга

if(b1 - b2 == 0)

{

double x1 = Math.Max(first.start.x, second.start.x);

double x2 = Math.Min(first.end.x, second.end.x);

double y1 = k1 \* x1 + b1;

double y2 = k1 \* x2 + b1;

return new Segment(x1, Math.Round(y1, 2), x2, Math.Round(y2, 2));

}

return noIntersection;

}

}

}

Файл Lab1Tests.cs:

using Lab1;

namespace Lab1\_Tests

{

[TestClass]

public class Lab1Tests

{

[TestMethod]

//Точка пересечения при паралельных отрезках

public void ParalelSegments()

{

Segment first = new Segment(0, 0, 5, 0);

Segment second = new Segment(0, 5, 5, 5);

Segment result = SegmentIntersection.Intersection(first, second);

Segment expected = new Segment(double.NaN, double.NaN, double.NaN, double.NaN);

Assert.AreEqual(expected, result);

}

[TestMethod]

//Область пересечения при накладывающихся отрезках

public void OverlappingSegments()

{

Segment first = new Segment(0, 0, 5, 5);

Segment second = new Segment(2, 2, 7, 7);

Segment result = SegmentIntersection.Intersection(first, second);

Segment expected = new Segment(2, 2, 5, 5);

Assert.AreEqual(expected, result);

}

[TestMethod]

//Точка пересечения, при одном вертикальном отрезке

public void OneVerticalSegment()

{

Segment first = new Segment(0, 0, 0, 5);

Segment second = new Segment(2, 2, 7, 7);

Segment result = SegmentIntersection.Intersection(first, second);

Segment expected = new Segment(double.NaN, double.NaN, double.NaN, double.NaN);

Assert.AreEqual(expected, result);

}

[TestMethod]

//Точка пересечения в общем случае

public void IntersectingSegments()

{

Segment first = new Segment(-5, -5, 5, 5);

Segment second = new Segment(-5, 5, 5, -5);

Segment result = SegmentIntersection.Intersection(first, second);

Segment expected = new Segment(0, 0, double.NaN, double.NaN);

Assert.AreEqual(expected, result);

}

}

}