УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №2.1

по предмету

Основы алгоритмизации и программирования

Вариант 3

Выполнил:

Бетеня К.С.

Проверила:

Данилова Г.В.

Группа 351005

Минск 2023

Задание:

Многоугольник задан координатами своих вершин. Найти площадь этого многоугольника.

Код программы на **Delphi**:

Program lab1;

Uses

System.SysUtils;

Const

MINCOORDINATE = -1000.0;

MAXCOORDINATE = 1000.0;

MINNUMBEROFSIDES = 3;

MAXNUMBEROFSIDES = 20;

Var

CoordinateMatrix: Array Of Array Of Real;

Area, SlopeFactor, Slopefactor1, Slopefactor2, YInterception, YInterception1,

YInterception2, IntersectionPoint: Real;

NumberOfSides, LimitforAmount, Index, I, J, HighXY, JOnI: Integer;

IsCorrect, IsCorrectCoordinate, IsCorrectAll, IsCorrectPolygon,

IsCorrectPoints: Boolean;

Begin

//inicialization

Area := 0.0;

SlopeFactor := 0.0; //y = rx + b => slopefactor = r

YInterception := 0.0; //y = fx + b => slopefactor = b

SlopeFactor1 := 0.0;

YInterception1 := 0.0;

SlopeFactor2 := 0.0;

YInterception2 := 0.0;

IntersectionPoint := 0.0;

HighXY := 0;

JOnI := 0;

LimitForAmount := 0; //high in main block

Index := 0; //x(index) and y(index)

NumberOfSides := 0;

IsCorrect := False; //for input

IsCorrectCoordinate := True; //for coordinate cheack

IsCorrectAll := True; //for all block

IsCorrectPolygon := True; //for poligon

IsCorrectPoints := True; //for points cheack

Write('This program calculates the area of a polygon.', 'The number of sides

of the polygon is selected by the user.', #13#10,

'You also need to enter the coordinates of the polygon vertices.',

#13#10, #13#10, '\*The Gauss formula is used for calculations\*',

#13#10, #13#10, 'Restrictions:', #13#10#9, '1. The number of sides of a

polygon', ' is an integer from 3 to 20;', #13#10#9,

'2. Coordinates - floating point', ' numbers from -1000.0 to 1000.0;',

#13#10#9, '3. All points must be unique (not ', 'repeated);',

#13#10#9, '4. The vertices of the polygon should be listed', ' in

traversal order (clockwise / counterclockwise).', #13#10, #13#10);

//input number of sides

Repeat

Write('Write number of sides of a polygon:', #13#10);

//cheack "Numeric input"

Try

Readln(NumberOfSides);

Except

Write('Error.', #13#10);

End;

//cheack restrictions

If (NumberOfSides < MINNUMBEROFSIDES) Or

(NumberOfSides > MAXNUMBEROFSIDES) Then

Write('The number of sides of a polygon is an integer from ',

MINNUMBEROFSIDES, ' to ', MAXNUMBEROFSIDES,

'. Try again.', #13#10)

Else

IsCorrect := True;

Until IsCorrect;

//cin x and y

Setlength(CoordinateMatrix, NumberOfSides, 2);

Repeat

IsCorrectAll := True;

HighXY := NumberOfSides - 1;

For I := 0 To HighXY Do

Begin

Index := I + 1;

IsCorrectCoordinate := False;

Repeat

//cin x

IsCorrect := False;

Repeat

Write('Write x', Index, ':', #13#10);

Try

Readln(CoordinateMatrix[I][0]);

IsCorrect := True;

Except

Write('Error. Try again', #13#10);

End;

Until IsCorrect;

//cin y

IsCorrect := False;

Repeat

Write('Write y', Index, ':', #13#10);

Try

Readln(CoordinateMatrix[I][1]);

IsCorrect := True;

Except

Write('Error. Try again.', #13#10);

End;

Until IsCorrect;

//we check the points to see if they are on the same line

If (I > 1) And (CoordinateMatrix[I - 1][0] - CoordinateMatrix[I –

2][0] <> 0) Then

Begin

If (CoordinateMatrix[I - 1][0] - CoordinateMatrix[I - 2][0] =

0) Then

Else

Begin

SlopeFactor := (CoordinateMatrix[I - 1][1] –

CoordinateMatrix[I - 2][1]) /

(CoordinateMatrix[I - 1][0] –

CoordinateMatrix[I – 2][0]);

YInterception := CoordinateMatrix[I - 1][1] –

CoordinateMatrix[I - 1][0] \* SlopeFactor;

If (CoordinateMatrix[I][1] = SlopeFactor \*

CoordinateMatrix[I][0] + YInterception) Then

Write('Three points cannot be on the same line. Try

again.', #13#10)

Else

IsCorrectCoordinate := True;

End;

End

Else

IsCorrectCoordinate := True;

Until IsCorrectCoordinate;

End;

//cheack that points cannot be repeated

For I := 0 To HighXY Do

Begin

JOnI := I + 1;

For J := JOnI To HighXY Do

Begin

If IsCorrectPoints And (CoordinateMatrix[I][0] =

CoordinateMatrix[J][0]) And

(CoordinateMatrix[I][1] = CoordinateMatrix[J][1]) Then

Begin

Write('Points must be unique. Try again.', #13#10);

IsCorrectAll := False;

IsCorrectPoints := False;

End;

End;

End;

//the main block of checking that there are no self-intersections

For I := 1 To HighXY Do

Begin

JOnI := I + 2;

If CoordinateMatrix[I][0] - CoordinateMatrix[I - 1][0] = 0 Then

Begin

YInterception1 := CoordinateMatrix[I][0];

For J := JOnI To HighXY Do

Begin

If CoordinateMatrix[J][0] - CoordinateMatrix[J - 1][0] = 0

Then

Begin

YInterception2 := CoordinateMatrix[J][0];

If (YInterception1 = YInterception2) Then

Begin

If ((((CoordinateMatrix[J][1] >

CoordinateMatrix[I][1]) And

(CoordinateMatrix[J][1] > CoordinateMatrix

[I - 1][1])) And ((CoordinateMatrix[J - 1][1] >

CoordinateMatrix[I][1]) And

(CoordinateMatrix[J - 1][1] > CoordinateMatrix

[I – 1][1]))) Or

(((CoordinateMatrix[J][1] <

CoordinateMatrix[I][1]) And

(CoordinateMatrix[J][1] < CoordinateMatrix

[I - 1][1])) And ((CoordinateMatrix[J - 1][1] <

CoordinateMatrix[I][1]) And

(CoordinateMatrix[J - 1][1] < CoordinateMatrix

[I - 1][1])))) Then

Else

IsCorrectPolygon := False;

End

End

Else

Begin

SlopeFactor2 := (CoordinateMatrix[J][1] –

CoordinateMatrix[J - 1][1]) /

(CoordinateMatrix[J][0] –

CoordinateMatrix[J - 1][0]);

YInterception2 := CoordinateMatrix[J][1] –

CoordinateMatrix[J][0] \* SlopeFactor2;

IntersectionPoint := (YInterception2 - YInterception1) /

(SlopeFactor1 - SlopeFactor2);

If (((YInterception1 > CoordinateMatrix[J][0]) And

(YInterception1 < CoordinateMatrix[J - 1][0])) Or

((YInterception1 < CoordinateMatrix[J][0]) And

(YInterception1 > CoordinateMatrix[J - 1][0])) And

(((IntersectionPoint > CoordinateMatrix[J][1]) And

(IntersectionPoint < CoordinateMatrix[J - 1][1])) Or

((IntersectionPoint < CoordinateMatrix[J][1]) And

(IntersectionPoint > CoordinateMatrix[J - 1][1]))))

Then

IsCorrectPolygon := False;

End;

End;

End

Else

If CoordinateMatrix[I][1] - CoordinateMatrix[I - 1][1] = 0 Then

Begin

YInterception1 := CoordinateMatrix[I][1];

For J := JOnI To HighXY Do

Begin

If CoordinateMatrix[J][0] - CoordinateMatrix[J - 1][0] = 0

Then

Begin

YInterception2 := CoordinateMatrix[I][1];

If (YInterception1 = YInterception2) Then

Begin

If ((((CoordinateMatrix[I][0] <

CoordinateMatrix[J][0]) And

(CoordinateMatrix[I][0] < CoordinateMatrix

[J - 1][0])) And

((CoordinateMatrix[I - 1][0] <

CoordinateMatrix[J][0]) And

(CoordinateMatrix[I - 1][0] <

CoordinateMatrix[J - 1][0]))) Or

(((CoordinateMatrix[I][0] >

CoordinateMatrix[J][0]) And

(CoordinateMatrix[I][0] > CoordinateMatrix

[J - 1][0])) And

((CoordinateMatrix[I - 1][0] >

CoordinateMatrix[J][0]) And

(CoordinateMatrix[I - 1][0] >

CoordinateMatrix[J - 1][0])))) Then

Else

IsCorrectPolygon := False;

End;

End

Else

Begin

SlopeFactor2 := (CoordinateMatrix[J][1] –

CoordinateMatrix[J - 1][1]) /

(CoordinateMatrix[J][0] –

CoordinateMatrix[J - 1][0]);

YInterception2 := CoordinateMatrix[J][1] –

CoordinateMatrix[J][0] \*

SlopeFactor2;

IntersectionPoint := (YInterception2 - YInterception1)

/ (SlopeFactor1 - SlopeFactor2);

If ((((YInterception1 > CoordinateMatrix[J][1]) And

(YInterception1 < CoordinateMatrix[J - 1][1])) Or

((YInterception1 < CoordinateMatrix[J][1]) And

(YInterception1 > CoordinateMatrix[J - 1][1])))

And (((IntersectionPoint > CoordinateMatrix[J][0])

And (IntersectionPoint < CoordinateMatrix

[J - 1][0])) Or ((IntersectionPoint <

CoordinateMatrix[J][0]) And (IntersectionPoint >

CoordinateMatrix[J - 1][0])))) Then

IsCorrectPolygon := False;

End;

End;

End

Else

Begin

SlopeFactor1 := (CoordinateMatrix[I][1] – CoordinateMatrix

[I - 1][1]) /

(CoordinateMatrix[I][0] - CoordinateMatrix[I - 1][0]);

YInterception1 := CoordinateMatrix[I][1] –

CoordinateMatrix[I][0] \* SlopeFactor1;

For J := JOnI To HighXY Do

Begin

If (CoordinateMatrix[J][0] –

CoordinateMatrix[J - 1][0] = 0) Then

Begin

YInterception2 := CoordinateMatrix[J][0];

IntersectionPoint := SlopeFactor1 \* YInterception2 +

YInterception1;

If (((IntersectionPoint > CoordinateMatrix[J][1]) And

(IntersectionPoint > CoordinateMatrix[J - 1][1]))

Or ((IntersectionPoint < CoordinateMatrix[J][1])

And (IntersectionPoint < CoordinateMatrix

[J - 1][1]))) Then

Else

IsCorrectPolygon := False;

End

Else

Begin

SlopeFactor2 := (CoordinateMatrix[J][1] –

CoordinateMatrix[J - 1][1]) /

(CoordinateMatrix[J][0] –

CoordinateMatrix[J - 1][0]);

YInterception2 := CoordinateMatrix[J][1] –

CoordinateMatrix[J][0] \*

SlopeFactor2;

IntersectionPoint := (YInterception2 - YInterception1)

/ (SlopeFactor1 - SlopeFactor2);

If ((((IntersectionPoint > CoordinateMatrix[J][0]) And

(IntersectionPoint < CoordinateMatrix[J - 1][0]))

Or ((IntersectionPoint < CoordinateMatrix[J][0])

And (IntersectionPoint > CoordinateMatrix

[J - 1][0]))) And (CoordinateMatrix[I][0] –

CoordinateMatrix[I - 1][0] =

CoordinateMatrix[J][0] – CoordinateMatrix

[J - 1][0]) And (CoordinateMatrix[I][1] –

CoordinateMatrix[I - 1][1] =

CoordinateMatrix[J][1] – CoordinateMatrix

[J - 1][1])) Then

IsCorrectPolygon := False;

End;

End;

End;

End;

//determine the test result

If (IsCorrectPolygon <> True) Then

Begin

IsCorrectAll := False;

IsCorrectPolygon := True;

Write('The rectangle must not be self-intersecting. Try again.',

#13#10);

End;

Until IsCorrectAll;

//main block

//we consider the result to be the Gauss formula

LimitForAmount := NumberOfSides - 2;

For I := 0 To LimitForAmount Do

Begin

//we calculate two amounts at once, taking into account the sign

Area := Area + (CoordinateMatrix[I][0] \* CoordinateMatrix

[I + 1][1]) - (CoordinateMatrix[I + 1][0] \*

CoordinateMatrix[I][1]);

End;

//transfer half the modulus of the available amount

Area := Abs(Area + (CoordinateMatrix[NumberOfSides - 1][0] \*

CoordinateMatrix[0][1]) - (CoordinateMatrix[NumberOfSides –

1][1] \* CoordinateMatrix[0][0]));

Area := Area / 2;

//cout resoult

Write(#13#10, 'Your area is: ', Area:7:3, '.', #13#10);

Writeln('Press any key to continue.');

Readln;

End.

Код программы на **C++**:

#include <iostream>

#include <iomanip> // for setprecision

int main()

{

const float MINCOORDINATE = -1000.0, MAXCOORDINATE = 1000.0;

float area, slopeFactor, slopeFactor1, slopeFactor2, yInterception,

yInterception1, yInterception2, intersectionPoint;

const int MINNUMBEROFSIDES = 3, MAXNUMBEROFSIDES = 20;

int numberOfSides, limitForAmount, index;

bool isInCorrect, isInCorrectCoordinate, isInCorrectAll, isInCorrectPolygon,

isCorrectPoints;

//inicialization

area = 0.0;

slopeFactor = 0.0;//y = rx + b => slopefactor = r

yInterception = 0.0;//y = fx + b => slopefactor = b

slopeFactor1 = 0.0;

yInterception1 = 0.0;

slopeFactor2 = 0.0;

yInterception2 = 0.0;

intersectionPoint = 0.0;

limitForAmount = 0;//high in main block

index = 0;//x(index) and y(index)

numberOfSides = 0;

isInCorrect = true;//for input

isInCorrectCoordinate = true;//for coordinate cheack

isInCorrectAll = false;//for all block

isInCorrectPolygon = false;//for poligon

isCorrectPoints = true;//for points cheack

//information about task

std::cout << "This program calculates the area of\ a polygon. The number of

sides of the polygon is selected by the user.\n"

"You also need to enter the coordinates of the polygon

vertices.\n\n"

"\*The Gauss formula is used for calculations\*\n\n"

"Restrictions: \n\t1. The number of sides of a polygon is an

integer from 3 to 20;\n"

"\t2. Coordinates - floating point numbers from -1000.0 to

1000.0;\n"

"\t3. All points must be unique (not repeated);\n"

"\t4. The vertices of the polygon should be listed in traversal

order (clockwise / counterclockwise).\n\n";

//formatted output

std::cout << std::setprecision(3) << std::fixed;

//input number of sides

do

{

std::cout << "Write number of sides of a polygon:\n";

std::cin >> numberOfSides;

//cheack "Numeric input"

if (std::cin.get() != '\n')

{

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "Error. Try again.\n";

}

//cheack restrictions

else if (numberOfSides < MINNUMBEROFSIDES || numberOfSides >

MAXNUMBEROFSIDES)

{

std::cout << "Error. The number of sides of a polygon is an integer

from " << MINNUMBEROFSIDES << " to " << MAXNUMBEROFSIDES

<< ". Try again.\n";

}

else

isInCorrect = false;

} while (isInCorrect);

//cin x and y

float\*\* coordinateMatrix = new float\*[numberOfSides];

do

{

isInCorrectAll = false;

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

{

index = i + 1;

coordinateMatrix[i] = new float[2];

isInCorrectCoordinate = true;

do

{

isInCorrect = true;

//cin x

do

{

std::cout << "Write x" << index << ":\n";

std::cin >> coordinateMatrix[i][0];

if (std::cin.get() != '\n')

{

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "Error. Try again.\n";

}

else

isInCorrect = false;

} while (isInCorrect);

//cin y

isInCorrect = true;

do

{

std::cout << "Write y" << index << ":\n";

std::cin >> coordinateMatrix[i][1];

if (std::cin.get() != '\n')

{

std::cin.clear();

std::cin.ignore(30000, '\n');

std::cout << "Error. Try again.\n";

}

else

isInCorrect = false;

} while (isInCorrect);

// we check the points to see if they are on the same line

if (i > 1 && coordinateMatrix[i - 1][0] - coordinateMatrix[i –

2][0] != 0)

{

if (coordinateMatrix[i - 1][0] - coordinateMatrix[i - 2][0]

== 0)

{

}

else

{

slopeFactor = (coordinateMatrix[i - 1][1] –

coordinateMatrix[i - 2][1]) /

(coordinateMatrix[i - 1][0] –

coordinateMatrix[i - 2][0]);

yInterception = coordinateMatrix[i - 1][1] –

coordinateMatrix[i - 1][0] \* slopeFactor;

if (coordinateMatrix[i][1] == slopeFactor \*

coordinateMatrix[i][0] + yInterception)

{

std::cout << "Three points cannot be on the same line.

Try again.\n";

}

else

isInCorrectCoordinate = false;

}

}

else

isInCorrectCoordinate = false;

} while (isInCorrectCoordinate);

}

// check that points cannot be repeated

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

{

for (int j = i + 1; j < numberOfSides; j++)

{

if (isCorrectPoints && coordinateMatrix[i][0] ==

coordinateMatrix[j][0] && coordinateMatrix[i][1] ==

coordinateMatrix[j][1])

{

std::cout << "Points must be unique. Try again.\n";

isInCorrectAll = true;

isCorrectPoints = false;

}

}

}

// the main block of checking that there are no self-intersections

for (int i = 1; i < numberOfSides; i++)

{

if (coordinateMatrix[i][0] - coordinateMatrix[i - 1][0] == 0)

{

yInterception1 = coordinateMatrix[i][0];

for (int j = i + 2; j < numberOfSides; j++)

{

if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0] == 0)

{

yInterception2 = coordinateMatrix[j][0];

if (yInterception1 == yInterception2)

{

if (((coordinateMatrix[j][1] > coordinateMatrix[i][1]

&& coordinateMatrix[j][1] > coordinateMatrix[i –

1][1]) && (coordinateMatrix[j - 1][1] >

coordinateMatrix[i][1] && coordinateMatrix[j –

1][1] > coordinateMatrix[i - 1][1])) ||

((coordinateMatrix[j][1] < coordinateMatrix[i][1]

&& coordinateMatrix[j][1] < coordinateMatrix[i –

1][1]) && (coordinateMatrix[j - 1][1] <

coordinateMatrix[i][1] && coordinateMatrix[j –

1][1] < coordinateMatrix[i - 1][1])))

{

}

else

isInCorrectPolygon = true;

}

}

else

{

slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);

yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \* slopeFactor2;

intersectionPoint = (yInterception2 - yInterception1) /

(slopeFactor1 - slopeFactor2);

if ((yInterception1 > coordinateMatrix[j][0] &&

yInterception1 < coordinateMatrix[j - 1][0]) ||

(yInterception1 < coordinateMatrix[j][0] &&

yInterception1 > coordinateMatrix[j - 1][0]) &&

((intersectionPoint > coordinateMatrix[j][1] &&

intersectionPoint < coordinateMatrix[j - 1][1]) ||

(intersectionPoint < coordinateMatrix[j][1] &&

intersectionPoint > coordinateMatrix[j - 1][1])))

isInCorrectPolygon = true;

}

}

}

else if (coordinateMatrix[i][1] - coordinateMatrix[i - 1][1] == 0)

{

yInterception1 = coordinateMatrix[i][1];

for (int j = i + 2; j < numberOfSides; j++)

{

if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0] == 0)

{

yInterception2 = coordinateMatrix[i][1];

if (yInterception1 == yInterception2)

{

if (((coordinateMatrix[i][0] < coordinateMatrix[j][0]

&& coordinateMatrix[i][0] < coordinateMatrix[j-1][0])

&& (coordinateMatrix[i-1][0] < coordinateMatrix[j][0]

&& coordinateMatrix[i - 1][0] < coordinateMatrix[j-

1][0])) || ((coordinateMatrix[i][0] >

coordinateMatrix[j][0] && coordinateMatrix[i][0] >

coordinateMatrix[j-1][0]) && (coordinateMatrix[i-1][0]

> coordinateMatrix[j][0] && coordinateMatrix[i - 1][0]

> coordinateMatrix[j-1][0])))

{

}

else

isInCorrectPolygon = true;

}

}

else

{

slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);

yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \* slopeFactor2;

intersectionPoint = (yInterception2 - yInterception1) /

(slopeFactor1 - slopeFactor2);

if (((yInterception1 > coordinateMatrix[j][1] &&

yInterception1 < coordinateMatrix[j - 1][1]) ||

(yInterception1 < coordinateMatrix[j][1] &&

yInterception1 > coordinateMatrix[j - 1][1])) &&

((intersectionPoint > coordinateMatrix[j][0] &&

intersectionPoint < coordinateMatrix[j - 1][0]) ||

(intersectionPoint < coordinateMatrix[j][0] &&

intersectionPoint > coordinateMatrix[j - 1][0])))

isInCorrectPolygon = true;

}

}

}

else

{

slopeFactor1 = (coordinateMatrix[i][1] - coordinateMatrix[i –

1][1]) / (coordinateMatrix[i][0] –

coordinateMatrix[i - 1][0]);

yInterception1 = coordinateMatrix[i][1] - coordinateMatrix[i][0] \*

slopeFactor1;

for (int j = i + 2; j < numberOfSides; j++)

{

if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0] == 0)

{

yInterception2 = coordinateMatrix[j][0];

intersectionPoint = slopeFactor1 \* yInterception2 +

yInterception1;

if ((intersectionPoint > coordinateMatrix[j][1] &&

intersectionPoint > coordinateMatrix[j - 1][1]) ||

(intersectionPoint < coordinateMatrix[j][1] &&

intersectionPoint < coordinateMatrix[j - 1][1]))

{

}

else

isInCorrectPolygon = true;

}

else

{

slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);

yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \* slopeFactor2;

intersectionPoint = (yInterception2 - yInterception1) /

(slopeFactor1 - slopeFactor2);

if (((intersectionPoint > coordinateMatrix[j][0] &&

intersectionPoint < coordinateMatrix[j - 1][0]) ||

(intersectionPoint < coordinateMatrix[j][0] &&

intersectionPoint > coordinateMatrix[j - 1][0])) &&

(coordinateMatrix[i][0] - coordinateMatrix[i - 1][0]

== coordinateMatrix[j][0] - coordinateMatrix[j –

1][0]) && (coordinateMatrix[i][1] –

coordinateMatrix[i - 1][1] == coordinateMatrix[j][1]

- coordinateMatrix[j - 1][1]))

isInCorrectPolygon = true;

}

}

}

}

// determine the test result

if (isInCorrectPolygon)

{

isInCorrectAll = true;

isInCorrectPolygon = false;

std::cout << "The rectangle must not be self-intersecting. Try

again.\n";

}

} while (isInCorrectAll);

// main block

// we consider the result to be the Gauss formula

limitForAmount = numberOfSides - 1;

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

{

// we calculate two amounts at once, taking into account the sign

area = area + (coordinateMatrix[i][0] \* coordinateMatrix[i +

1][1]) - (coordinateMatrix[i + 1][0] \*

coordinateMatrix[i][1]);

}

// transfer half the modulus of the available amount

area = abs(area + (coordinateMatrix[numberOfSides - 1][0] \*

coordinateMatrix[0][1]) - (coordinateMatrix[numberOfSides –

1][1] \* coordinateMatrix[0][0]));

area = area / 2;

// cout resoult

std::cout << "\nYour area is: " << area << ".\n";

//cleaning the memory

for (int i = 0; i < numberOfSides; i++) {

delete[] coordinateMatrix[i];

coordinateMatrix[i] = nullptr;

}

delete[] coordinateMatrix;

coordinateMatrix = nullptr;

return 0;

}

Код программы на **Java**:

package lab2;  
  
import java.util.Scanner;  
  
import static java.lang.Math.*abs*;  
  
public class lab1 {  
 public static void main(String[] args)  
 {  
 Scanner in = new Scanner(System.*in*);  
  
 final float MINCOORDINATE = (float) -1000.0,  
 MAXCOORDINATE = (float) 1000.0;  
 float area, slopeFactor, slopeFactor1, slopeFactor2, yInterception,  
 yInterception1, yInterception2, intersectionPoint;  
 final int MINNUMBEROFSIDES = 3, MAXNUMBEROFSIDES = 20;  
 int numberOfSides, limitForAmount, index, jOnI;  
 boolean isInCorrect, isInCorrectCoordinate, isInCorrectAll, isInCorrectPolygon,  
 isCorrectPoints;  
  
 //inicialization  
 area = 0.0F;  
 slopeFactor = 0.0F;//y = rx + b => slopefactor = r  
 yInterception = 0.0F;//y = fx + b => slopefactor = b  
 slopeFactor1 = 0.0F;  
 yInterception1 = 0.0F;  
 slopeFactor2 = 0.0F;  
 yInterception2 = 0.0F;  
 intersectionPoint = 0.0F;  
 limitForAmount = 0;//high in main block  
 index = 0;//x(index) and y(index)  
 numberOfSides = 0;  
 jOnI = 0;  
 isInCorrect = true;//for input  
 isInCorrectCoordinate = true;//for coordinate cheack  
 isInCorrectAll = false;//for all block  
 isInCorrectPolygon = false;//for poligon  
 isCorrectPoints = true;//for points cheack  
  
 System.*out*.print("""  
 This program calculates the area of a polygon. The

number of sides of the polygon is selected by the

user.  
 You also need to enter the coordinates of the polygon

vertices.  
   
 \*The Gauss formula is used for calculations\*  
   
 Restrictions:\s  
 1. The number of sides of a polygon is an

integer from 3 to 20;  
 2. Coordinates - floating point numbers from

–1000.0 to 1000.0;  
 3. All points must be unique (not repeated);  
 4. The vertices of the polygon should be

listed in traversal order (clockwise /

counterclockwise).  
   
 """);  
  
 //input number of sides  
 do  
 {  
 System.*out*.print("Write number of sides of a polygon:\n");  
 //cheack "Numeric input"  
 try  
 {  
 numberOfSides = Short.*parseShort*(in.nextLine());  
 }  
 catch (NumberFormatException error)  
 {  
 System.*err*.print("Error.\n");  
 }  
 //cheack restrictions  
 if (numberOfSides < MINNUMBEROFSIDES || numberOfSides >

MAXNUMBEROFSIDES)  
 {  
 System.*err*.printf("The number of sides of a polygon is an integer

from %d to %d. Try again.\n", MINNUMBEROFSIDES,

MAXNUMBEROFSIDES);  
 }  
 else  
 isInCorrect = false;  
 } while (isInCorrect);  
  
 //cin x and y  
 float[][] coordinateMatrix = new float[numberOfSides][2];  
 do  
 {  
 isInCorrectAll = false;  
 for (int i = 0; i < numberOfSides; i++)  
 {  
 index = i + 1;  
 isInCorrectCoordinate = true;  
 do  
 {  
 isInCorrect = true;  
 //cin x  
 do  
 {  
 System.*out*.printf("Write x%d:\n", index);  
 try  
 {  
 coordinateMatrix[i][0] =

Float.*parseFloat*(in.nextLine());  
 isInCorrect = false;  
 } catch (NumberFormatException error)  
 {  
 System.*out*.print("Error. Try again.\n");  
 }  
 } while (isInCorrect);  
 //cin y  
 isInCorrect = true;  
 do  
 {  
 System.*out*.printf("Write y%d:\n", index);  
 try  
 {  
 coordinateMatrix[i][1] =

Float.*parseFloat*(in.nextLine());  
 isInCorrect = false;  
 }  
 catch (NumberFormatException error)  
 {  
 System.*out*.print("Error. Try again.\n");  
 }  
 } while (isInCorrect);  
  
 // we check the points to see if they are on the same line

if (i > 1 && coordinateMatrix[i - 1][0] - coordinateMatrix[i –

2][0] != 0)

{

if (coordinateMatrix[i - 1][0] - coordinateMatrix[i –

2][0] == 0)

{

}

else

{

slopeFactor = (coordinateMatrix[i - 1][1] –

coordinateMatrix[i - 2][1]) /

(coordinateMatrix[i - 1][0] –

coordinateMatrix[i - 2][0]);

yInterception = coordinateMatrix[i - 1][1] –

coordinateMatrix[i - 1][0] \*

slopeFactor;

if (coordinateMatrix[i][1] == slopeFactor \*

coordinateMatrix[i][0] + yInterception)

{

std::cout << "Three points cannot be on the same

line. Try again.\n";

}

else

isInCorrectCoordinate = false;

}

}

else

isInCorrectCoordinate = false;  
 } while (isInCorrectCoordinate);  
 }  
 // check that points cannot be repeated  
 for (int i = 0; i < numberOfSides; i++)  
 {  
 jOnI = i + 1;  
 for (int j = jOnI; j < numberOfSides; j++)  
 {  
 if (isCorrectPoints && coordinateMatrix[i][0] ==

coordinateMatrix[j][0] && coordinateMatrix[i][1] ==

coordinateMatrix[j][1])  
 {  
 System.*out*.print("Points must be unique. Try again.\n");  
 isInCorrectAll = true;  
 isCorrectPoints = false;  
 }  
 }  
 }  
  
 // the main block of checking that there are no self-intersections  
 for (int i = 1; i < numberOfSides; i++)  
 {  
 if (coordinateMatrix[i][0] - coordinateMatrix[i - 1][0] == 0)  
 {  
 yInterception1 = coordinateMatrix[i][0];  
 for (int j = i + 2; j < numberOfSides; j++)  
 {  
 if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0]

== 0)  
 {  
 yInterception2 = coordinateMatrix[j][0];  
 if (yInterception1 == yInterception2)  
 {  
 if (((coordinateMatrix[j][1] >

coordinateMatrix[i][1] &&

coordinateMatrix[j][1] > coordinateMatrix[i –

1][1]) && (coordinateMatrix[j - 1][1] >

coordinateMatrix[i][1] && coordinateMatrix[j –

1][1] > coordinateMatrix[i - 1][1])) ||  
 ((coordinateMatrix[j][1] <

coordinateMatrix[i][1] &&

coordinateMatrix[j][1] < coordinateMatrix[i –

1][1]) && (coordinateMatrix[j - 1][1] <

coordinateMatrix[i][1] && coordinateMatrix[j –

1][1] < coordinateMatrix[i - 1][1])))  
 {  
  
 }  
 else  
 isInCorrectPolygon = true;  
 }  
 }  
 else  
 {  
 slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);  
 yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \*

slopeFactor2;  
 intersectionPoint = (yInterception2 - yInterception1)

/ (slopeFactor1 - slopeFactor2);  
 if ((yInterception1 > coordinateMatrix[j][0] &&

yInterception1 < coordinateMatrix[j - 1][0]) ||  
 (yInterception1 < coordinateMatrix[j][0] &&

yInterception1 > coordinateMatrix[j - 1][0]) &&  
 ((intersectionPoint > coordinateMatrix[j][1] &&

intersectionPoint < coordinateMatrix[j - 1][1]) ||  
 (intersectionPoint < coordinateMatrix[j][1] &&

intersectionPoint > coordinateMatrix[j - 1][1])))  
 isInCorrectPolygon = true;  
 }  
 }  
 }  
 else if (coordinateMatrix[i][1] - coordinateMatrix[i - 1][1] == 0)  
 {  
 yInterception1 = coordinateMatrix[i][1];  
 for (int j = i + 2; j < numberOfSides; j++)  
 {  
 if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0]

== 0)  
 {  
 yInterception2 = coordinateMatrix[i][1];  
 if (yInterception1 == yInterception2)  
 {  
 if (((coordinateMatrix[i][0] <

coordinateMatrix[j][0] &&

coordinateMatrix[i][0] < coordinateMatrix[j-

1][0]) && (coordinateMatrix[i-1][0] <

coordinateMatrix[j][0] && coordinateMatrix[i –

1][0] < coordinateMatrix[j-1][0])) ||  
 ((coordinateMatrix[i][0] >

coordinateMatrix[j][0] &&

coordinateMatrix[i][0] > coordinateMatrix[j-

1][0]) && (coordinateMatrix[i-1][0] >

coordinateMatrix[j][0] && coordinateMatrix[i –

1][0] > coordinateMatrix[j-1][0])))  
 {  
  
 }  
 else  
 isInCorrectPolygon = true;  
 }  
 }  
 else  
 {  
 slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);  
 yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \*

slopeFactor2;  
 intersectionPoint = (yInterception2 - yInterception1)

/ (slopeFactor1 - slopeFactor2);  
 if (((yInterception1 > coordinateMatrix[j][1] &&

yInterception1 < coordinateMatrix[j - 1][1]) ||  
 (yInterception1 < coordinateMatrix[j][1] &&

yInterception1 > coordinateMatrix[j - 1][1])) &&  
 ((intersectionPoint > coordinateMatrix[j][0] &&

intersectionPoint < coordinateMatrix[j - 1][0]) ||  
 (intersectionPoint < coordinateMatrix[j][0] &&

intersectionPoint > coordinateMatrix[j - 1][0])))  
 isInCorrectPolygon = true;  
 }  
 }  
 }  
 else  
 {  
 slopeFactor1 = (coordinateMatrix[i][1] - coordinateMatrix[i –

1][1]) / (coordinateMatrix[i][0] –

coordinateMatrix[i - 1][0]);  
 yInterception1 = coordinateMatrix[i][1] –

coordinateMatrix[i][0] \* slopeFactor1;  
 for (int j = i + 2; j < numberOfSides; j++)  
 {  
 if (coordinateMatrix[j][0] - coordinateMatrix[j - 1][0]

== 0)  
 {  
 yInterception2 = coordinateMatrix[j][0];  
 intersectionPoint = slopeFactor1 \* yInterception2 +

yInterception1;  
 if ((intersectionPoint > coordinateMatrix[j][1] &&

intersectionPoint > coordinateMatrix[j - 1][1]) ||  
 (intersectionPoint < coordinateMatrix[j][1] &&

intersectionPoint < coordinateMatrix[j - 1][1]))  
 {  
  
 }  
 else  
 isInCorrectPolygon = true;  
 }  
 else  
 {  
 slopeFactor2 = (coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]) /

(coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]);  
 yInterception2 = coordinateMatrix[j][1] –

coordinateMatrix[j][0] \*

slopeFactor2;  
 intersectionPoint = (yInterception2 - yInterception1)

/ (slopeFactor1 - slopeFactor2);  
 if (((intersectionPoint > coordinateMatrix[j][0] &&

intersectionPoint < coordinateMatrix[j - 1][0]) ||  
 (intersectionPoint < coordinateMatrix[j][0] &&

intersectionPoint > coordinateMatrix[j - 1][0]))

&& (coordinateMatrix[i][0] - coordinateMatrix[i –

1][0] == coordinateMatrix[j][0] –

coordinateMatrix[j - 1][0]) &&  
 (coordinateMatrix[i][1] - coordinateMatrix[i –

1][1] == coordinateMatrix[j][1] –

coordinateMatrix[j - 1][1]))  
 isInCorrectPolygon = true;  
 }  
 }  
 }  
 }  
  
 // determine the test result  
 if (isInCorrectPolygon)  
 {  
 isInCorrectAll = true;  
 isInCorrectPolygon = false;  
 System.*out*.print("The rectangle must not be self-intersecting. Try

again.\n");  
 }  
 in.close();  
 } while (isInCorrectAll);

// main block  
 // we consider the result to be the Gauss formula  
 limitForAmount = (short) (numberOfSides - 1);  
 for (int i = 0; i < limitForAmount; i++)  
 {  
 // we calculate two amounts at once, taking into account the

sign (+/-)  
 area = area + (coordinateMatrix[i][0] \* coordinateMatrix[i +

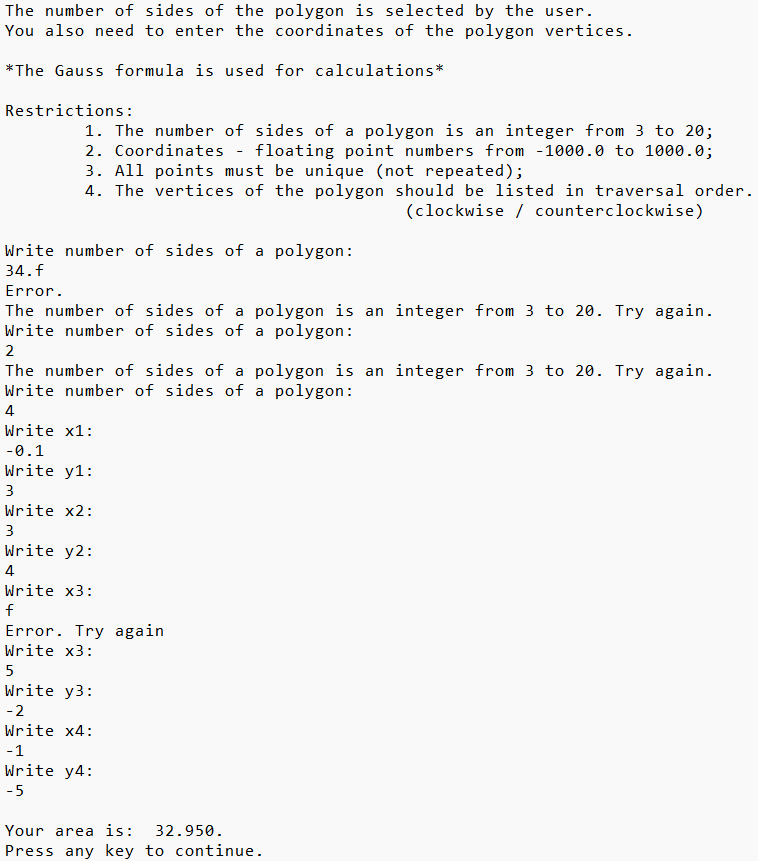
1][1]) - (coordinateMatrix[i + 1][0] \*

coordinateMatrix[i][1]);  
 }  
 // transfer half the modulus of the available amount  
 area = *abs*(area + (coordinateMatrix[numberOfSides - 1][0] \*

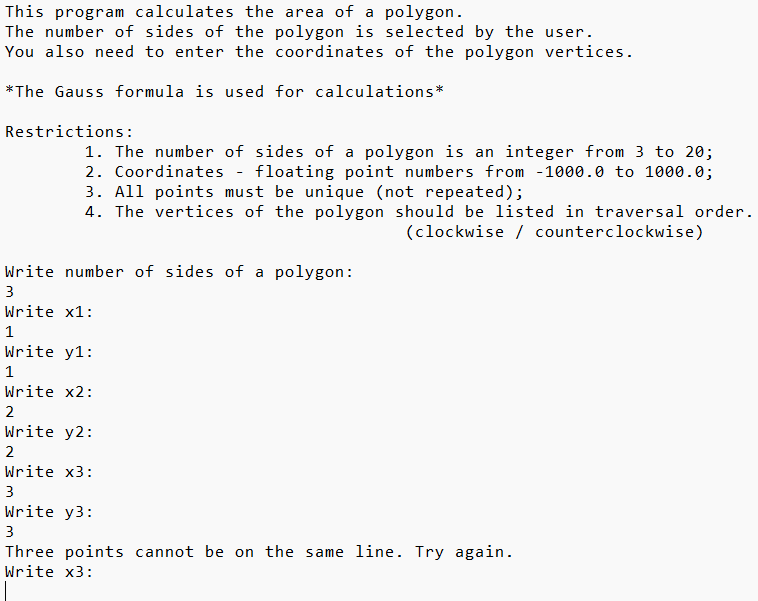
coordinateMatrix[0][1]) - (coordinateMatrix[numberOfSides –

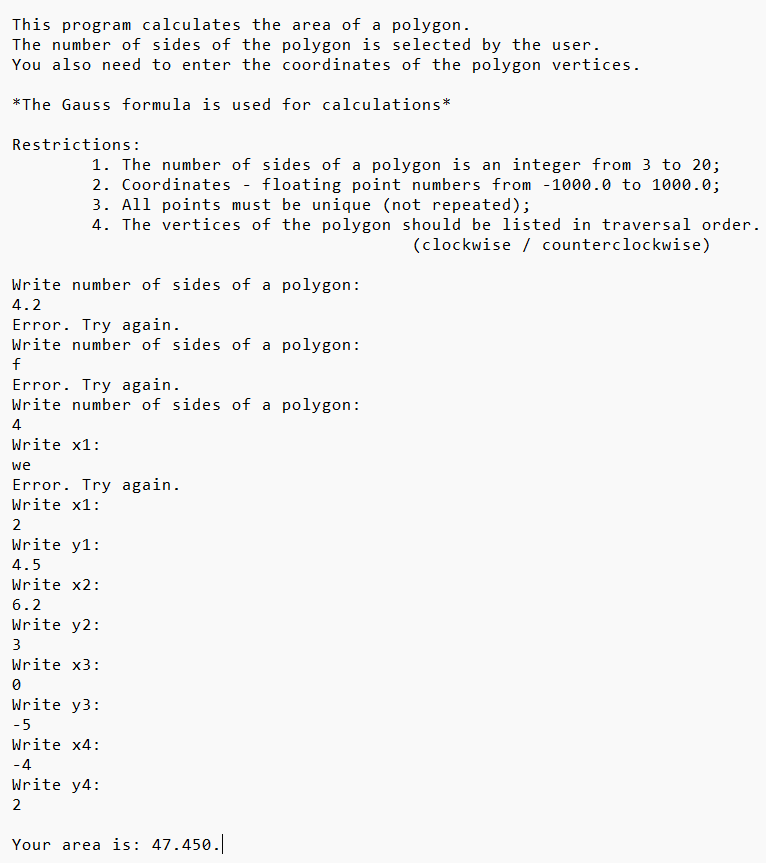
1][1] \* coordinateMatrix[0][0]));  
 area = area / 2;  
 // cout resoult  
 System.*out*.printf("\nYour area is: %.3f.\n", area);  
 }  
}

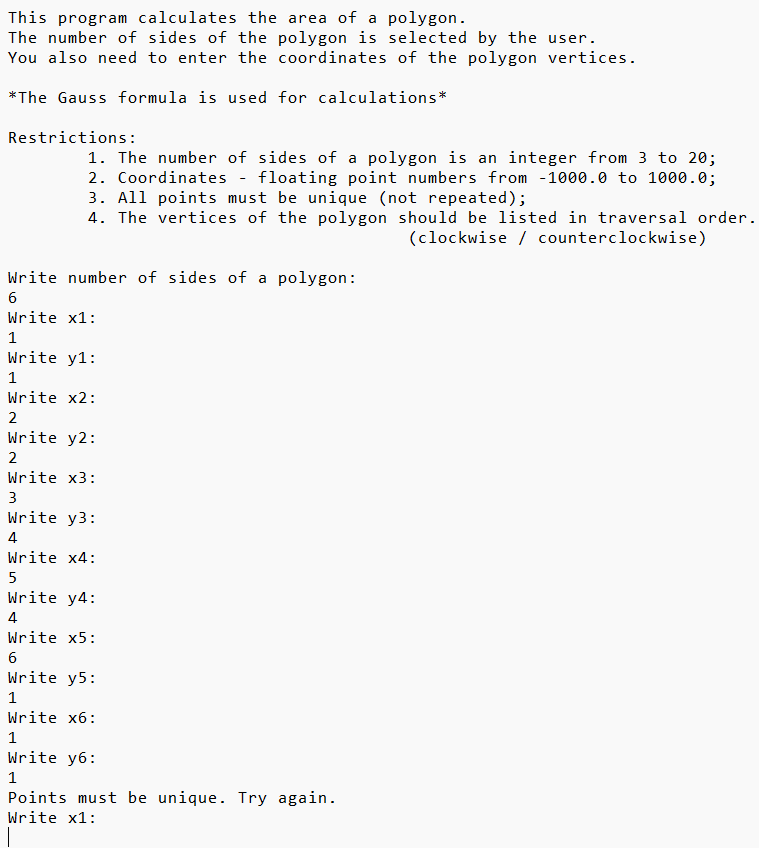
Результат на **Delphi**:



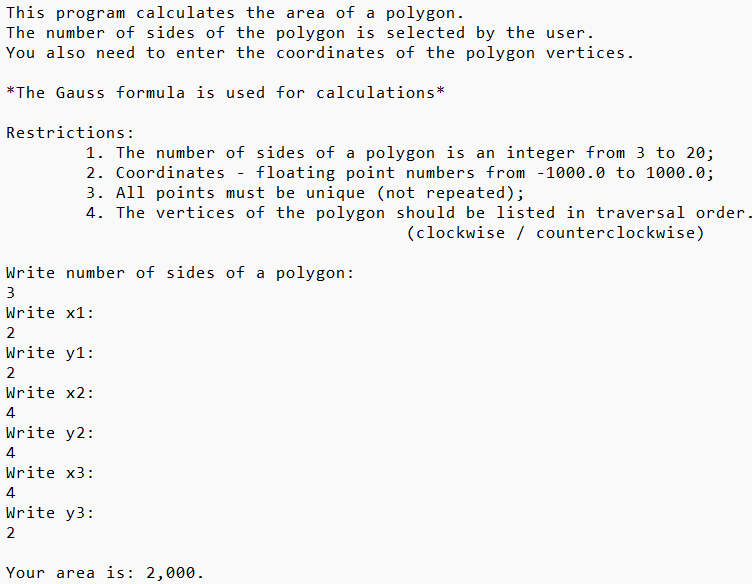
Результат на **C++**:

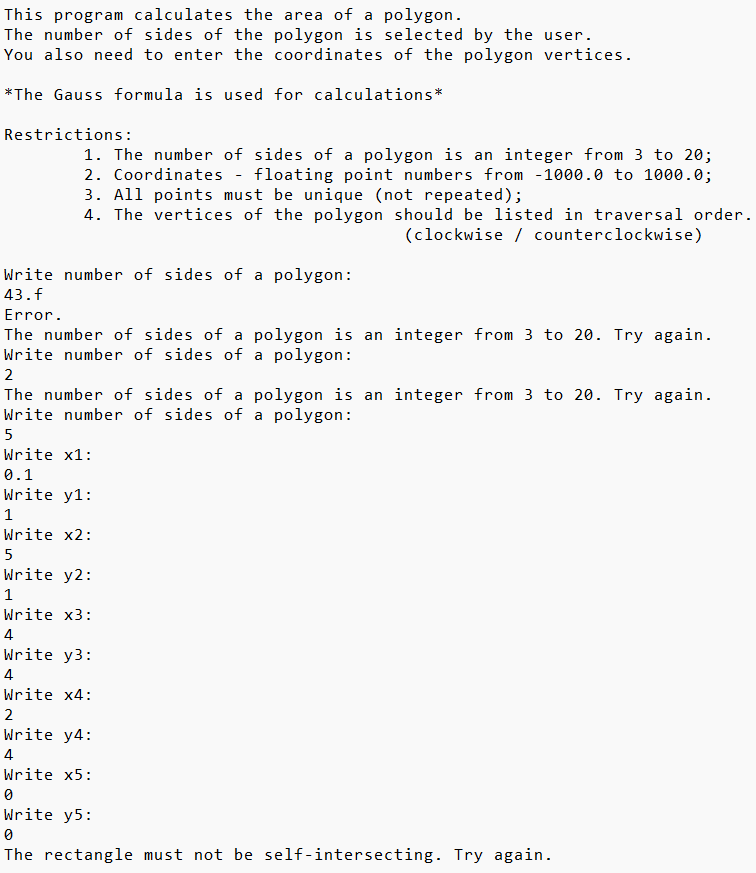






Результат на **Java**:





Блок-схема:

