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

MINSIDES = 3;

MAXSIDES = 20;

Var

CoordMat: Array Of Array Of Real;

Area, SlpFact, SlpFact1, SlpFact2, YInter, YInter1, YInter2, IntPoint: Real;

SidesNumb, LimForAmt, Ind, I, J, HighXY, JOnI: Integer;

IsCorrect: Boolean;

Begin

//inicialization

Area := 0.0;

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

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

SlpFact1 := 0.0;

YInter1 := 0.0;

SlpFact2 := 0.0;

YInter2 := 0.0;

IntPoint := 0.0;

HighXY := 0;

JOnI := 0;

LimForAmt := 0; //high in main block

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

SidesNumb := 0;

IsCorrect := False; //for input

Write('This program calculates the area of a polygon.', #13#10, '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.', #13#10#9#9#9#9#9,

'(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(SidesNumb);

Except

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

End;

//cheack restrictions

If (SidesNumb < MINSIDES) Or (SidesNumb > MAXSIDES) Then

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

MINSIDES, ' to ', MAXSIDES, '. Try again.', #13#10)

Else

IsCorrect := True;

Until IsCorrect;

//cin x and y

Setlength(CoordMat, SidesNumb, 2);

Repeat

HighXY := SidesNumb - 1;

For I := 0 To HighXY Do

Begin

Ind := I + 1;

IsCorrect := False;

Repeat

//cin x

IsCorrect := False;

Repeat

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

Try

Readln(CoordMat[I][0]);

IsCorrect := True;

Except

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

End;

Until IsCorrect;

//cin y

IsCorrect := False;

Repeat

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

Try

Readln(CoordMat[I][1]);

IsCorrect := True;

Except

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

End;

Until IsCorrect;

IsCorrect := False;

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

If (I > 1) And (CoordMat[I - 1][0] - CoordMat[I - 2][0] <> 0) Then

Begin

If (CoordMat[I - 1][0] - CoordMat[I - 2][0] = 0) Then

Else

Begin

SlpFact := (CoordMat[I - 1][1] - CoordMat[I - 2][1]) /

(CoordMat[I - 1][0] - CoordMat[I - 2][0]);

YInter := CoordMat[I - 1][1] - CoordMat[I - 1][0] \*

SlpFact;

If (CoordMat[I][1] = SlpFact \* CoordMat[I][0] + YInter)

Then

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

again.', #13#10)

Else

IsCorrect := True;

End;

End

Else

IsCorrect := True;

Until IsCorrect;

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 IsCorrect = False And (CoordMat[I][0] = CoordMat[J][0]) And

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

Begin

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

IsCorrect := True;

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 CoordMat[I][0] - CoordMat[I - 1][0] = 0 Then

Begin

YInter1 := CoordMat[I][0];

For J := JOnI To HighXY Do

Begin

If CoordMat[J][0] - CoordMat[J - 1][0] = 0 Then

Begin

YInter2 := CoordMat[J][0];

If (YInter1 = YInter2) Then

Begin

If ((((CoordMat[J][1] > CoordMat[I][1]) And

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

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

(CoordMat[J - 1][1] > CoordMat[I - 1][1]))) Or

(((CoordMat[J][1] < CoordMat[I][1]) And

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

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

(CoordMat[J - 1][1] < CoordMat[I - 1][1])))) Then

Else

IsCorrect := False;

End

End

Else

Begin

SlpFact2 := (CoordMat[J][1] - CoordMat[J - 1][1]) /

(CoordMat[J][0] - CoordMat[J - 1][0]);

YInter2 := CoordMat[J][1] - CoordMat[J][0] \* SlpFact2;

IntPoint := (YInter2 - YInter1) / (SlpFact1 - SlpFact2);

If (((YInter1 > CoordMat[J][0]) And (YInter1 <

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

((YInter1 < CoordMat[J][0]) And (YInter1 >

CoordMat[J - 1][0])) And

(((IntPoint > CoordMat[J][1]) And (IntPoint <

CoordMat[J - 1][1])) Or ((IntPoint < CoordMat[J][1])

And (IntPoint > CoordMat[J - 1][1])))) Then

IsCorrect := False;

End;

End;

End

Else

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

Begin

YInter1 := CoordMat[I][1];

For J := JOnI To HighXY Do

Begin

If CoordMat[J][0] - CoordMat[J - 1][0] = 0 Then

Begin

YInter2 := CoordMat[I][1];

If (YInter1 = YInter2) Then

Begin

If ((((CoordMat[I][0] < CoordMat[J][0]) And

(CoordMat[I][0] < CoordMat[J - 1][0])) And

((CoordMat[I - 1][0] < CoordMat[J][0]) And

(CoordMat[I - 1][0] < CoordMat[J - 1][0]))) Or

(((CoordMat[I][0] > CoordMat[J][0]) And

(CoordMat[I][0] > CoordMat[J - 1][0])) And

((CoordMat[I - 1][0] > CoordMat[J][0]) And

(CoordMat[I - 1][0] > CoordMat[J - 1][0]))))

Then

Else

IsCorrect := False;

End;

End

Else

Begin

SlpFact2 := (CoordMat[J][1] - CoordMat[J - 1][1]) /

(CoordMat[J][0] - CoordMat[J - 1][0]);

YInter2 := CoordMat[J][1] - CoordMat[J][0] \* SlpFact2;

IntPoint := (YInter2 - YInter1) / (SlpFact1 –

SlpFact2);

If ((((YInter1 > CoordMat[J][1]) And (YInter1 <

CoordMat[J - 1][1])) Or ((YInter1 <

CoordMat[J][1]) And (YInter1 > CoordMat[J –

1][1]))) And (((IntPoint > CoordMat[J][0]) And

(IntPoint < CoordMat[J - 1][0])) Or

((IntPoint < CoordMat[J][0]) And (IntPoint >

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

IsCorrect := False;

End;

End;

End

Else

Begin

SlpFact1 := (CoordMat[I][1] - CoordMat[I - 1][1]) /

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

YInter1 := CoordMat[I][1] - CoordMat[I][0] \* SlpFact1;

For J := JOnI To HighXY Do

Begin

If (CoordMat[J][0] - CoordMat[J - 1][0] = 0) Then

Begin

YInter2 := CoordMat[J][0];

IntPoint := SlpFact1 \* YInter2 + YInter1;

If (((IntPoint > CoordMat[J][1]) And (IntPoint >

CoordMat[J - 1][1])) Or

((IntPoint < CoordMat[J][1]) And (IntPoint <

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

Else

IsCorrect := False;

End

Else

Begin

SlpFact2 := (CoordMat[J][1] - CoordMat[J - 1][1]) /

(CoordMat[J][0] - CoordMat[J - 1][0]);

YInter2 := CoordMat[J][1] - CoordMat[J][0] \* SlpFact2;

IntPoint := (YInter2 - YInter1) / (SlpFact1 –

SlpFact2);

If ((((IntPoint > CoordMat[J][0]) And (IntPoint <

CoordMat[J - 1][0])) Or ((IntPoint <

CoordMat[J][0]) And (IntPoint > CoordMat[J –

1][0]))) And (CoordMat[I][0] - CoordMat[I - 1][0]

= CoordMat[J][0] - CoordMat[J - 1][0]) And

(CoordMat[I][1] - CoordMat[I - 1][1] =

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

IsCorrect := False;

End;

End;

End;

End;

//determine the test result

If (IsCorrect <> True) Then

Begin

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

#13#10);

End;

Until IsCorrect;

//main block

//we consider the result to be the Gauss formula

LimForAmt := SidesNumb - 2;

For I := 0 To LimForAmt Do

Begin

//we calculate two amounts at once, taking into account the sign (+/-)

Area := Area + (CoordMat[I][0] \* CoordMat[I + 1][1]) - (CoordMat[I + 1][0]

\* CoordMat[I][1]);

End;

//transfer half the modulus of the available amount

Area := Abs(Area + (CoordMat[SidesNumb - 1][0] \* CoordMat[0][1]) –

(CoordMat[SidesNumb - 1][1] \* CoordMat[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 int MINSIDES = 3, MAXSIDES = 20;

float area, slpFact, slpFact1, slpFact2, yInter, yInter1, yInter2, intPoint;

int sidesNumb, limForAmt, ind, jOnI;

bool isInCorrect;

//inicialization

area = 0.0;

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

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

slpFact1 = 0.0;

yInter1 = 0.0;

slpFact2 = 0.0;

yInter2 = 0.0;

intPoint = 0.0;

limForAmt = 0;//high in main block

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

sidesNumb = 0;

jOnI = 0;

isInCorrect = true;//for input

//information about task

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

"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.\n"

"\t\t\t\t\t(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 >> sidesNumb;

//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 (sidesNumb < MINSIDES || sidesNumb > MAXSIDES)

{

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

from " << MINSIDES << " to " << MAXSIDES << ". Try

again.\n";

}

else

isInCorrect = false;

} while (isInCorrect);

//cin x and y

float\*\* coordMat = new float\*[sidesNumb];

do

{

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

{

ind = i + 1;

coordMat[i] = new float[2];

do

{

isInCorrect = true;

//cin x

do

{

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

std::cin >> coordMat[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" << ind << ":\n";

std::cin >> coordMat[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);

isInCorrect = true;

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

if (i > 1 && coordMat[i - 1][0] - coordMat[i - 2][0] != 0)

{

if (coordMat[i - 1][0] - coordMat[i - 2][0] == 0)

{

}

else

{

slpFact = (coordMat[i - 1][1] - coordMat[i - 2][1]) /

(coordMat[i - 1][0] - coordMat[i - 2][0]);

yInter = coordMat[i - 1][1] - coordMat[i - 1][0] \*

slpFact;

if (coordMat[i][1] == slpFact \* coordMat[i][0] + yInter)

{

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

Try again.\n";

}

else

isInCorrect = false;

}

}

else

isInCorrect = false;

} while (isInCorrect);

}

// check that points cannot be repeated

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

{

jOnI = i + 1;

for (int j = jOnI; j < sidesNumb; j++)

{

if (isInCorrect = false && coordMat[i][0] == coordMat[j][0] &&

coordMat[i][1] == coordMat[j][1])

{

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

isInCorrect = true;

}

}

}

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

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

{

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

{

yInter1 = coordMat[i][0];

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

{

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

{

yInter2 = coordMat[j][0];

if (yInter1 == yInter2)

{

if (((coordMat[j][1] > coordMat[i][1] &&

coordMat[j][1] > coordMat[i - 1][1]) &&

(coordMat[j - 1][1] > coordMat[i][1] && coordMat[j

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

((coordMat[j][1] < coordMat[i][1] &&

coordMat[j][1] < coordMat[i - 1][1]) &&

(coordMat[j - 1][1] < coordMat[i][1] && coordMat[j

- 1][1] < coordMat[i - 1][1])))

{

}

else

isInCorrect = true;

}

}

else

{

slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);

yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;

intPoint = (yInter2 - yInter1) / (slpFact1 - slpFact2);

if ((yInter1 > coordMat[j][0] && yInter1 < coordMat[j –

1][0]) || (yInter1 < coordMat[j][0] && yInter1 >

coordMat[j - 1][0]) && ((intPoint > coordMat[j][1] &&

intPoint < coordMat[j - 1][1]) || (intPoint <

coordMat[j][1] && intPoint > coordMat[j - 1][1])))

isInCorrect = true;

}

}

}

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

{

yInter1 = coordMat[i][1];

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

{

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

{

yInter2 = coordMat[i][1];

if (yInter1 == yInter2)

{

if (((coordMat[i][0] < coordMat[j][0] &&

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

(coordMat[i-1][0] < coordMat[j][0] && coordMat[i –

1][0] < coordMat[j-1][0])) ||

((coordMat[i][0] > coordMat[j][0] &&

coordMat[i][0] > coordMat[j-1][0]) &&

(coordMat[i-1][0] > coordMat[j][0] && coordMat[i –

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

{

}

else

isInCorrect = true;

}

}

else

{

slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);

yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;

intPoint = (yInter2 - yInter1) / (slpFact1 - slpFact2);

if (((yInter1 > coordMat[j][1] && yInter1 < coordMat[j –

1][1]) || (yInter1 < coordMat[j][1] && yInter1 >

coordMat[j - 1][1])) && ((intPoint > coordMat[j][0] &&

intPoint < coordMat[j - 1][0]) || (intPoint <

coordMat[j][0] && intPoint > coordMat[j - 1][0])))

isInCorrect = true;

}

}

}

else

{

slpFact1 = (coordMat[i][1] - coordMat[i - 1][1]) / (coordMat[i][0]

- coordMat[i - 1][0]);

yInter1 = coordMat[i][1] - coordMat[i][0] \* slpFact1;

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

{

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

{

yInter2 = coordMat[j][0];

intPoint = slpFact1 \* yInter2 + yInter1;

if ((intPoint > coordMat[j][1] && intPoint > coordMat[j –

1][1]) || (intPoint < coordMat[j][1] && intPoint <

coordMat[j - 1][1]))

{

}

else

isInCorrect = true;

}

else

{

slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);

yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;

intPoint = (yInter2 - yInter1) / (slpFact1 - slpFact2);

if (((intPoint > coordMat[j][0] && intPoint < coordMat[j –

1][0]) || (intPoint < coordMat[j][0] && intPoint >

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

1][0] == coordMat[j][0] - coordMat[j - 1][0]) &&

(coordMat[i][1] - coordMat[i - 1][1] == coordMat[j][1]

- coordMat[j - 1][1]))

isInCorrect = true;

}

}

}

}

// determine the test result

if (isInCorrect)

{

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

again.\n";

}

} while (isInCorrect);

// main block

// we consider the result to be the Gauss formula

limForAmt = sidesNumb - 1;

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

{

// we calculate two amounts at once, taking into account the sign (+/-)

area = area + (coordMat[i][0] \* coordMat[i + 1][1]) - (coordMat[i + 1][0]

\* coordMat[i][1]);

}

// transfer half the modulus of the available amount

area = abs(area + (coordMat[sidesNumb - 1][0] \* coordMat[0][1]) –

(coordMat[sidesNumb - 1][1] \* coordMat[0][0]));

area = area / 2;

// cout resoult

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

//cleaning the memory

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

delete[] coordMat[i];

coordMat[i] = nullptr;

}

delete[] coordMat;

coordMat = 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 int MINSIDES = 3,  
 MAXSIDES = 20;  
 float area, slpFact, slpFact1, slpFact2, yInter,  
 yInter1, yInter2, intPoint;  
 int sidesNumb, limForAmt, ind, jOnI;  
 boolean isInCorrect;  
  
 //inicialization  
 area = 0.0F;  
 slpFact = 0.0F;//y = rx + b => slopefactor = r  
 yInter = 0.0F;//y = fx + b => slopefactor = b  
 slpFact1 = 0.0F;  
 yInter1 = 0.0F;  
 slpFact2 = 0.0F;  
 yInter2 = 0.0F;  
 intPoint = 0.0F;  
 limForAmt = 0;//high in main block  
 ind = 0;//x(index) and y(index)  
 sidesNumb = 0;  
 jOnI = 0;  
 isInCorrect = true;//for input  
  
 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  
 {  
 sidesNumb = Short.*parseShort*(in.nextLine());  
 }  
 catch (NumberFormatException error)  
 {  
 System.*err*.print("Error.\n");  
 }  
 //cheack restrictions  
 if (sidesNumb < MINSIDES || sidesNumb > MAXSIDES)  
 {  
 System.*err*.printf("The number of sides of a polygon is an integer

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

MAXSIDES);  
 }  
 else  
 isInCorrect = false;  
 } while (isInCorrect);  
  
 //cin x and y  
 float[][] coordMat = new float[sidesNumb][2];  
 do  
 {  
 for (int i = 0; i < sidesNumb; i++)  
 {  
 ind = i + 1;  
 do  
 {  
 isInCorrect = true;  
 //cin x  
 do  
 {  
 System.*out*.printf("Write x%d:\n", ind);  
 try  
 {  
 coordMat[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", ind);  
 try  
 {  
 coordMat[i][1] = Float.*parseFloat*(in.nextLine());  
 isInCorrect = false;  
 }  
 catch (NumberFormatException error)  
 {  
 System.*out*.print("Error. Try again.\n");  
 }  
 } while (isInCorrect);  
  
 isInCorrect = true;  
 // we check the points to see if they are on the same line  
 if (i > 1)  
 {  
 if (coordMat[i - 1][0] - coordMat[i - 2][0] == 0)  
 {  
  
 }  
 else  
 {  
 slpFact = (coordMat[i - 1][1] - coordMat[i - 2][1]) /

(coordMat[i - 1][0] - coordMat[i - 2][0]);  
 yInter = coordMat[i - 1][1] - coordMat[i - 1][0] \*

slpFact;  
 if (coordMat[i][1] == slpFact \* coordMat[i][0] +

yInter)  
 {  
 System.*out*.print("Three points cannot be on the

same line. Try again.\n");  
 }  
 else  
 {  
 isInCorrect = false;  
 }  
 }  
 }  
 else  
 {  
 isInCorrect = false;  
 }  
 } while (isInCorrect);  
 }  
 // check that points cannot be repeated  
 for (int i = 0; i < sidesNumb; i++)  
 {  
 jOnI = i + 1;  
 for (int j = jOnI; j < sidesNumb; j++)  
 {  
 if (isInCorrect = false && coordMat[i][0] == coordMat[j][0] &&

coordMat[i][1] == coordMat[j][1])  
 {  
 System.*out*.print("Points must be unique. Try again.\n");  
 isInCorrect = true;  
 }  
 }  
 }  
  
 // the main block of checking that there are no self-intersections  
 for (int i = 1; i < sidesNumb; i++)  
 {  
 if (coordMat[i][0] - coordMat[i - 1][0] == 0)  
 {  
 yInter1 = coordMat[i][0];  
 for (int j = i + 2; j < sidesNumb; j++)  
 {  
 if (coordMat[j][0] - coordMat[j - 1][0] == 0)  
 {  
 yInter2 = coordMat[j][0];  
 if (yInter1 == yInter2)  
 {  
 if (((coordMat[j][1] > coordMat[i][1] &&

coordMat[j][1] > coordMat[i - 1][1]) &&  
 (coordMat[j - 1][1] > coordMat[i][1] &&

coordMat[j - 1][1] > coordMat[i - 1][1])) ||  
 ((coordMat[j][1] < coordMat[i][1] &&

coordMat[j][1] < coordMat[i - 1][1]) &&  
 (coordMat[j - 1][1] < coordMat[i][1] &&

coordMat[j - 1][1] < coordMat[i - 1][1])))  
 {  
  
 }  
 else  
 isInCorrect = true;  
 }  
 }  
 else  
 {  
 slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);  
 yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;  
 intPoint = (yInter2 - yInter1) / (slpFact1 –

slpFact2);  
 if ((yInter1 > coordMat[j][0] && yInter1 < coordMat[j

- 1][0]) || (yInter1 < coordMat[j][0] && yInter1 >

coordMat[j - 1][0]) && ((intPoint > coordMat[j][1]

&& intPoint < coordMat[j - 1][1]) || (intPoint <

coordMat[j][1] && intPoint > coordMat[j - 1][1])))  
 isInCorrect = true;  
 }  
 }  
 }  
 else if (coordMat[i][1] - coordMat[i - 1][1] == 0)  
 {  
 yInter1 = coordMat[i][1];  
 for (int j = i + 2; j < sidesNumb; j++)  
 {  
 if (coordMat[j][0] - coordMat[j - 1][0] == 0)  
 {  
 yInter2 = coordMat[i][1];  
 if (yInter1 == yInter2)  
 {  
 if (((coordMat[i][0] < coordMat[j][0] &&

coordMat[i][0] < coordMat[j-1][0]) &&  
 (coordMat[i-1][0] < coordMat[j][0] &&

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

coordMat[i][0] > coordMat[j-1][0]) &&  
 (coordMat[i-1][0] > coordMat[j][0] &&

coordMat[i - 1][0] > coordMat[j-1][0])))  
 {  
  
 }  
 else  
 isInCorrect = true;  
 }  
 }  
 else  
 {  
 slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);  
 yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;  
 intPoint = (yInter2 - yInter1) / (slpFact1 –

slpFact2);  
 if (((yInter1 > coordMat[j][1] && yInter1 < coordMat[j

- 1][1]) || (yInter1 < coordMat[j][1] && yInter1 >

coordMat[j - 1][1])) && ((intPoint >

coordMat[j][0] && intPoint < coordMat[j - 1][0])

|| (intPoint < coordMat[j][0] && intPoint >

coordMat[j - 1][0])))  
 isInCorrect = true;  
 }  
 }  
 }  
 else  
 {  
 slpFact1 = (coordMat[i][1] - coordMat[i - 1][1]) /

(coordMat[i][0] - coordMat[i - 1][0]);  
 yInter1 = coordMat[i][1] - coordMat[i][0] \* slpFact1;  
 for (int j = i + 2; j < sidesNumb; j++)  
 {  
 if (coordMat[j][0] - coordMat[j - 1][0] == 0)  
 {  
 yInter2 = coordMat[j][0];  
 intPoint = slpFact1 \* yInter2 + yInter1;  
 if ((intPoint > coordMat[j][1] && intPoint >

coordMat[j - 1][1]) ||  
 (intPoint < coordMat[j][1] && intPoint < coordMat[j –

1][1]))  
 {  
  
 }  
 else  
 isInCorrect = true;  
 }  
 else  
 {  
 slpFact2 = (coordMat[j][1] - coordMat[j - 1][1]) /

(coordMat[j][0] - coordMat[j - 1][0]);  
 yInter2 = coordMat[j][1] - coordMat[j][0] \* slpFact2;  
 intPoint = (yInter2 - yInter1) / (slpFact1 –

slpFact2);  
 if (((intPoint > coordMat[j][0] && intPoint <

coordMat[j - 1][0]) || (intPoint < coordMat[j][0]

&& intPoint > coordMat[j - 1][0])) &&

(coordMat[i][0] - coordMat[i - 1][0] ==

coordMat[j][0] - coordMat[j - 1][0]) &&

(coordMat[i][1] - coordMat[i - 1][1] ==

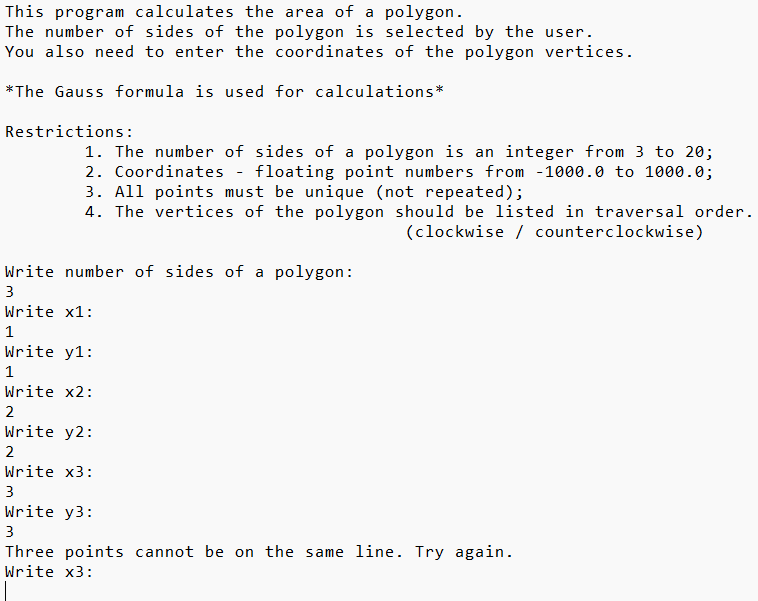
coordMat[j][1] - coordMat[j - 1][1]))  
 isInCorrect = true;  
 }  
 }  
 }  
 }  
 // determine the test result  
 if (isInCorrect)  
 {  
 System.*out*.print("The rectangle must not be self-intersecting. Try

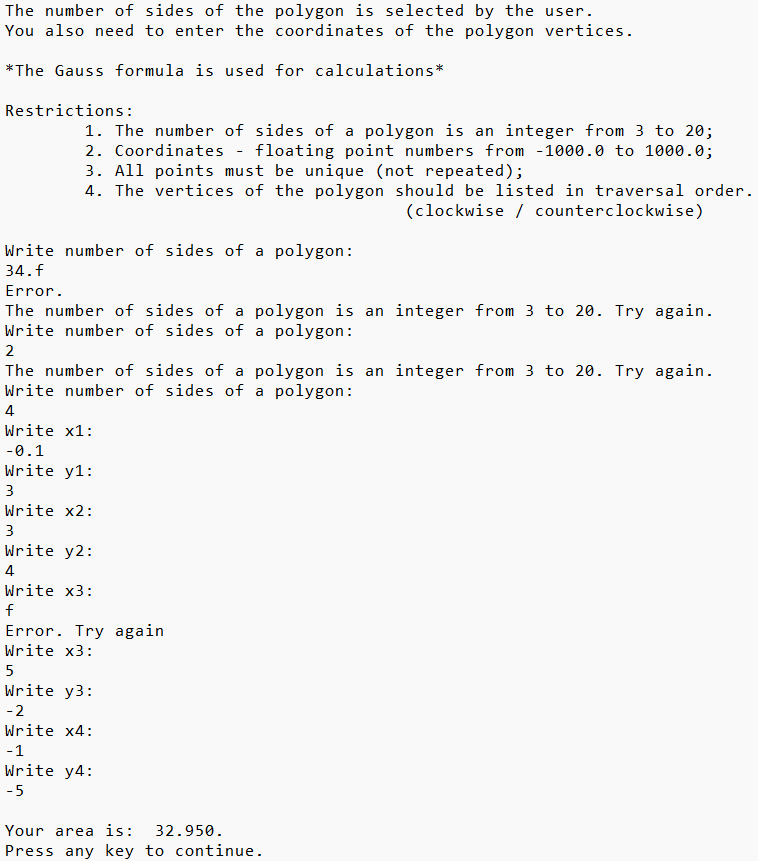
again.\n");  
 }  
 in.close();  
 } while (isInCorrect);  
 // main block  
 // we consider the result to be the Gauss formula  
 limForAmt = (short) (sidesNumb - 1);  
 for (int i = 0; i < limForAmt; i++)  
 {  
 // we calculate two amounts at once, taking into account the sign +/-  
 area = area + (coordMat[i][0] \* coordMat[i + 1][1]) - (coordMat[i +

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

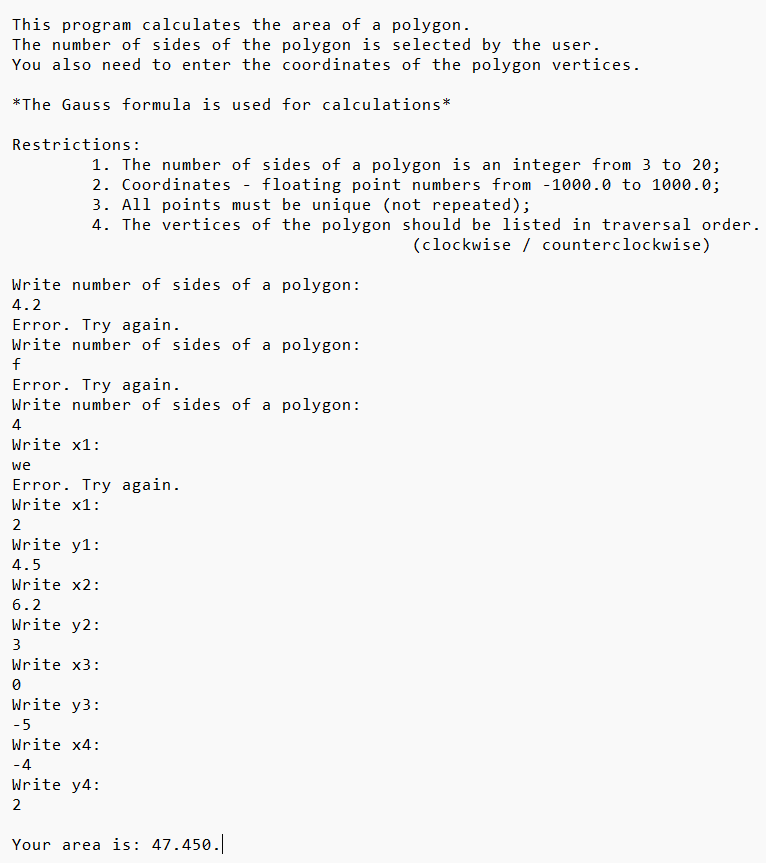
(coordMat[sidesNumb - 1][1] \* coordMat[0][0]));  
 area = area / 2;  
 // cout resoult  
 System.*out*.printf("\nYour area is: %.3f.\n", area);  
 }  
}

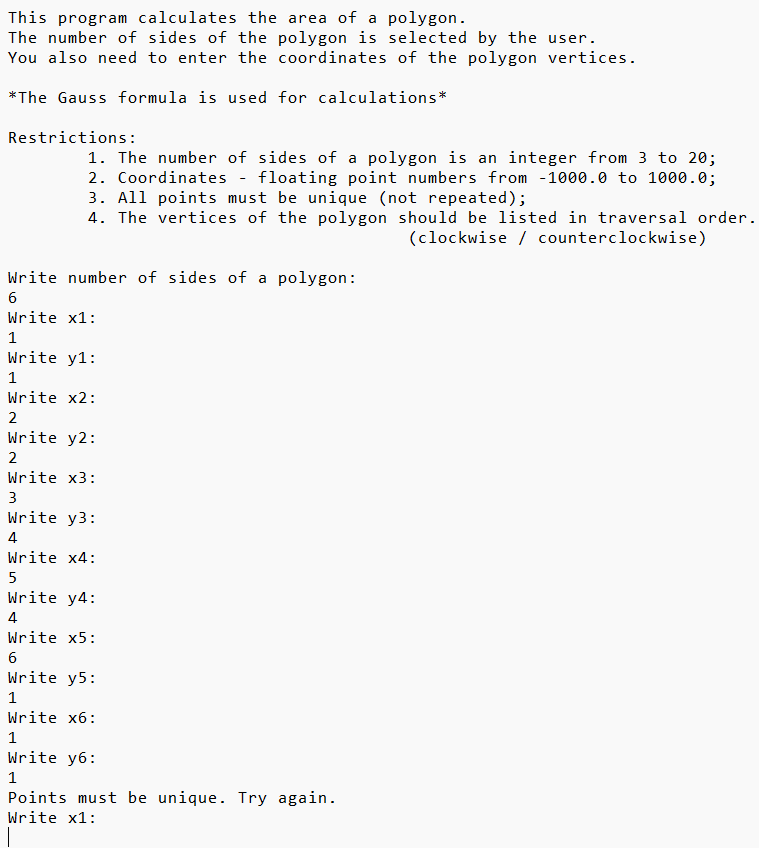
Результат на **Delphi**:



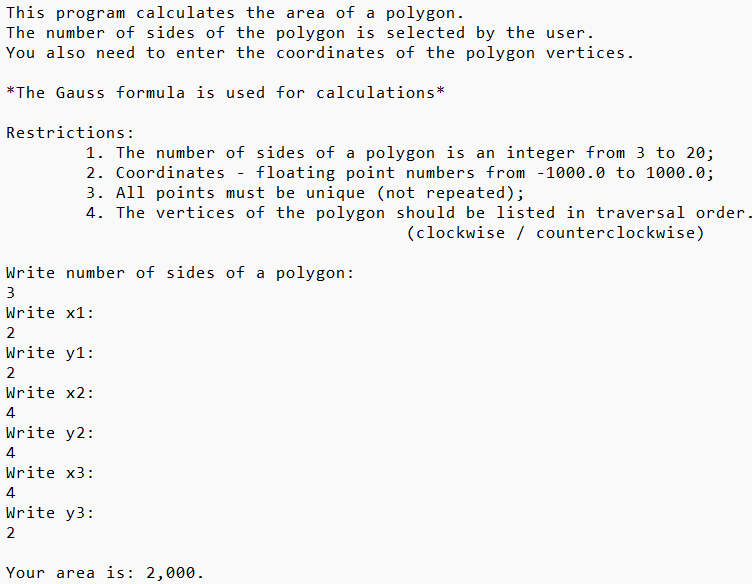


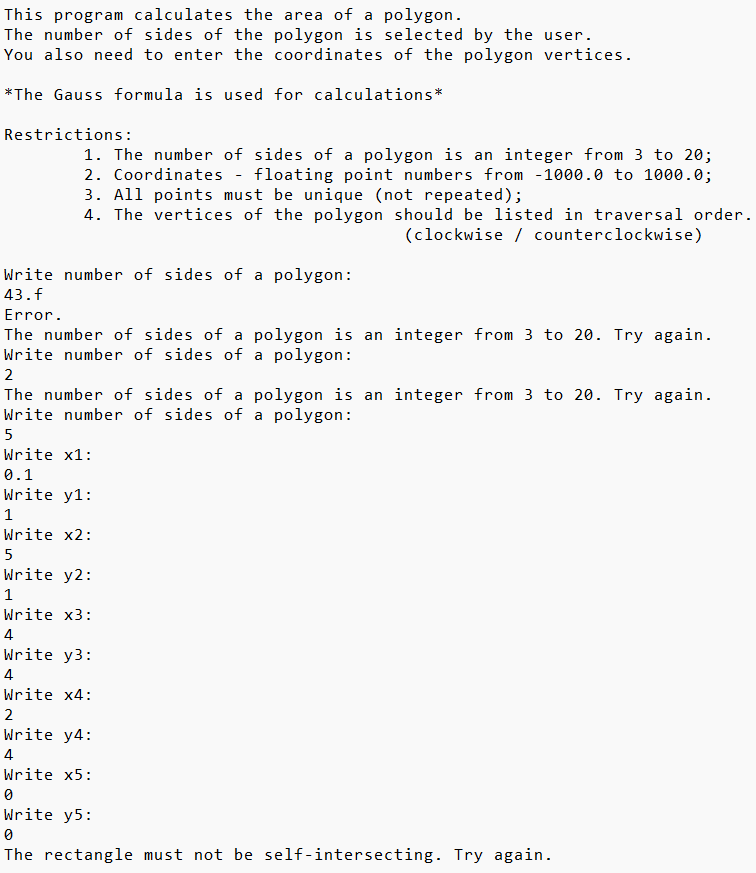
Результат на **C++**:





Результат на **Java**:





Блок-схема: