# КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ТАРАСА ШЕВЧЕНКА



## ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**Кафедра прикладних інформаційних систем**

**Звіт до лабораторної роботи №2**

# з курсу

**«Проектування та аналіз обчислювальних алгоритмів»**

*Студента 2 курсу*

*групи ПП-21 спеціальності 122 «Комп'ютерні науки» ОП «Прикладне програмування»*

%username%

*Викладач:*

к.ф.-м.н., доц. Шолохов О.В.

## Київ – 202

1. **Назва роботи**

Визначення приналежності довільної точки довільному опуклому багатокутнику

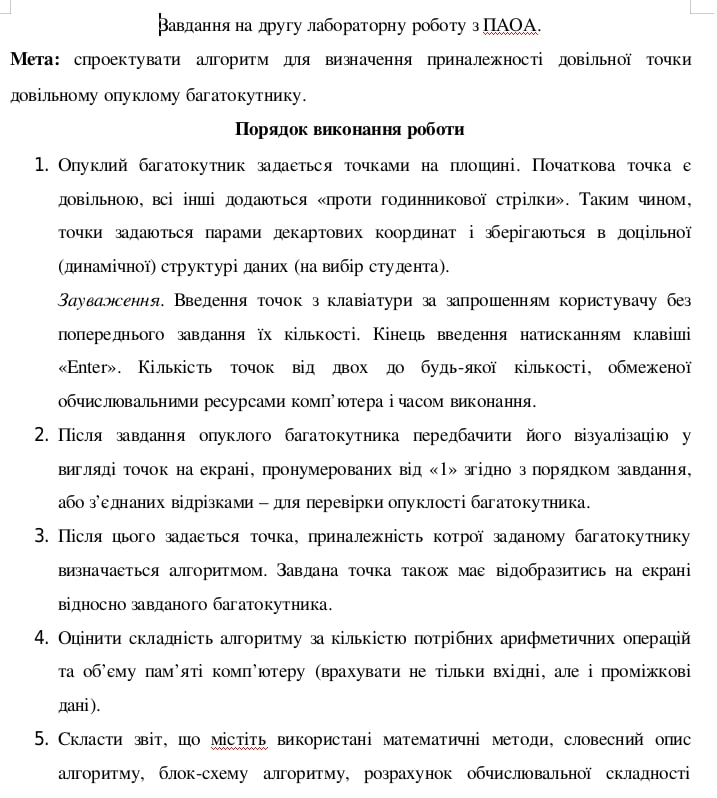
1. **Тема роботи**

Визначення приналежності довільної точки довільному опуклому багатокутнику

1. **Мета роботи**

Створити алгоритм для визначення приналежності довільної точки довільному опуклому багатокутнику у двовимірному просторі

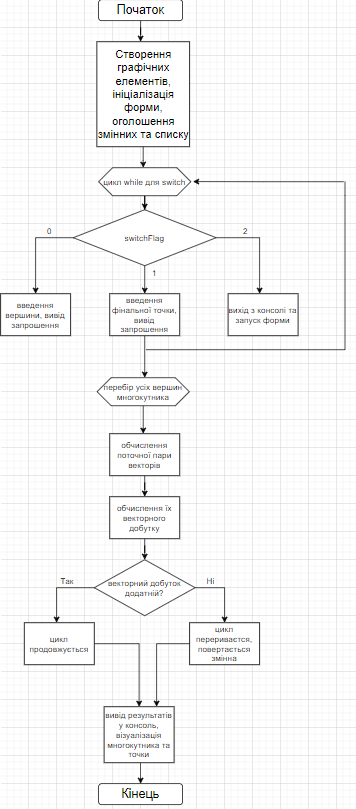
1. **Умова завдання**
2. Опуклий багатокутник задається точками на площині. Початкова точка є довільною, всі інші додаються проти годинникової стрілки. Таким чином, точки задаються парами декартових координат і зберігаються в доцільній динамічній структурі данних. (введення точок з клавіатури без попереднього задання їх кількості; кінець введення натисканням клавіші “Enter”; кількість точок від двох до довільної, обмеженої ресурсами комп’ютера та часом виконання)
3. Після задання опуклого багатокутника передбачити його візуалізацію у вигляді точок на екрані, пронумерованих або з’єднаних відрізками - для перевірки опуклості багатокутника.
4. Після цього задається точка, приналежність якої багатокутнику визначається алгоритмом. Задана точка теж має бути відображена на екрані.
5. Оцінити складність алгоритму за кількістю арифметичних операцій та об’єму пам’яті комп’ютера (врахувати не тільки вхідні, але й проміжкові дані)
6. Скласти звіт, що містить використані математичні методи, словесний опис та блок-схему алгоритму, розрахунок обчислювальної складності.



1. **Словесний опис алгоритму**

Створюємо графічні елементи, потрібні змінні та список точок, ініціалізуємо форму. Описуємо меню для введення точок, реалізоване за допомогою switch у циклі while: 0 для введення вершини многокутника, 1 для введення фінальної точки, 2 для запуску форми. Після цього описуємо алгоритм визначення приналежності точки багатокутнику, що приймає масив заданих вершин та задану точку, після чого обраховує кожну пару векторів (кожна пара являє собою ребро та вектор від вершини до заданої точки, причому останній завжди перший у парі) та їх векторний добуток. Згідно з теорією, якщо точка належить многокутнику, то при обчисленні вищезгаданим чином таких векторів їх векторні добутки будуть одного знаку (якщо вершини задані проти годинникової стрілки, то добутки додатні; якщо проти, то від’ємні). Відповідно, алгоритм перевіряє знак обрахованого добутку у циклі, і якщо добуток від’ємний, то цикл завершуєтся і алгоритм повертає булеву змінну isNegative зі значенням true, виводячи у консоль рядок “OUT!”. Якщо ж серед добутків від’ємних немає, то змінна повертаєтся зі значенням false, виводячи у консоль рядок “IN!”. Алгоритм, як і візуалізація многокутника та точки, викликається натисканням на кнопку START у нижньому правому куті застосунку.

Блок-схема:



Код:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Po\_209

{

public partial class Form1 : Form

{

Graphics gr; Pen pen; Point tempPoint; int switchFlag = 0; Point finalPoint;

LinkedList<Point> points = new LinkedList<Point>();

public Form1()

{

InitializeComponent();

gr = this.CreateGraphics();

pen = new Pen(Color.Black, 3);

this.DoubleBuffered = true;

while (switchFlag != 2)

{

switch (switchFlag)

{

case 0:

{

Console.WriteLine("Enter the point coordinate by coordinate with X being first:");

tempPoint.X = Convert.ToInt32(Console.ReadLine());

tempPoint.Y = Convert.ToInt32(Console.ReadLine());

points.AddLast(tempPoint);

Console.WriteLine("Enter 0 to continue entering points, 1 to enter final point or 2 to quit:");

switchFlag = Convert.ToInt32(Console.ReadLine());

break;

}

case 1:

{

Console.WriteLine("Enter the final point; X, then Y:");

finalPoint.X = Convert.ToInt32(Console.ReadLine());

finalPoint.Y = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Enter 2 to quit:");

switchFlag = Convert.ToInt32(Console.ReadLine());

break;

}

case 2:

{

break;

}

}

}

}

public bool vectorFinder(Point[] polygon)

{

LinkedList<Point> vectors = new LinkedList<Point>();

Point currentEdge = new Point();

Point currentVectorToPoint = new Point();

bool isNegative;

for (int q = 0; q < polygon.Length - 1; q++)

{

currentEdge.X = (polygon[q + 1].X - polygon[q].X);

currentEdge.Y = (polygon[q + 1].Y - polygon[q].Y);

currentVectorToPoint.X = (finalPoint.X - polygon[q].X);

currentVectorToPoint.Y = (finalPoint.Y - polygon[q].Y);

Console.WriteLine(Convert.ToString((currentVectorToPoint.X \* currentEdge.X) + (currentVectorToPoint.Y \* currentEdge.Y)));

if ((currentVectorToPoint.X \* currentEdge.X) + (currentVectorToPoint.Y \* currentEdge.Y) < 0 )

{

isNegative = true;

return isNegative;

} else

{

isNegative = false;

}

}

currentEdge.X = (polygon[0].X) - (polygon[polygon.Length - 1].X);

currentEdge.Y = (polygon[0].Y) - (polygon[polygon.Length - 1].Y);

currentVectorToPoint.X = (finalPoint.X) - (polygon[polygon.Length - 1].X);

currentVectorToPoint.Y = (finalPoint.Y) - (polygon[polygon.Length - 1].Y);

Console.WriteLine(Convert.ToString((currentVectorToPoint.X \* currentEdge.X) + (currentVectorToPoint.Y \* currentEdge.Y)));

if ((currentVectorToPoint.X \* currentEdge.X) + (currentVectorToPoint.Y \* currentEdge.Y) < 0)

{

isNegative = true;

}

else

{

isNegative = false;

}

return isNegative;

}

private void button2\_Click\_1(object sender, EventArgs e)

{

Point[] pointsArray = points.ToArray<Point>();

gr.DrawPolygon(pen, pointsArray);

System.Drawing.SolidBrush myBrush = new System.Drawing.SolidBrush(System.Drawing.Color.Red);

gr.DrawEllipse(pen, finalPoint.X, finalPoint.Y, 1, 1);

gr.DrawString("X", Font, myBrush, finalPoint.X, finalPoint.Y);

for (int q = 0; q < pointsArray.Length; q++)

{

gr.DrawString(Convert.ToString(q), Font, myBrush, pointsArray[q].X, pointsArray[q].Y);

}

if (vectorFinder(pointsArray))

{

Console.WriteLine("OUT!");

}

else

{

Console.WriteLine("IN!");

}

}

private void button3\_Click(object sender, EventArgs e)

{

pictureBox3.Visible = true;

}

}

}

1. **Складність алгоритму**

### Оскільки для визначення приналежності точки многокутнику нам потрібно обійти всі його вершини, то алгоритм лінійний зі складністю O(n). Программа займає 4 кілобайти, об’єм використаної пам’яті напряму залежить від введених даних; під час роботи в середньому программа використовує 9 мегабайт оперативної пам’яті.

Результат роботи програми:

[screenshot]

1. **Висновки**

В результаті виконання даної лабораторної роботи я створив алгоритм для виконання типових завдань на визначення приналежності точки опуклому многокутнику шляхом перевірки векторного добутку пар “грань - вектор з вершини до заданої точки”. Вважаю дану лабораторну роботу виконаною в повному обсязі.