# Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 4 з дисципліни

«Основи програмування-2.

Базові конструкції»

«Перевантаження операторів»

Варіант 20

Виконав студент Лошак Віктор Іванович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

( прізвище, ім'я, по батькові)

Київ 2022

**Лабораторна робота 4**

**Мета:**

Вивчити механізми створення класів з використанням перевантажених операторів(операцій).

**Тема:** Визначити клас «Трикутник» членами якого є сторони трикутника в просторі. Реалізувати для нього декілька конструкторів, геттери, методи обчислення площі трикутника. Перевантажити оператори: ++, --, +=, -=. Створити три трикутники( Т1, Т2, Т3) використовуючи різні конструктори. Інкрементувати довжини сторін трикутника Т1, а довжини сторін Т2 декрементувати. Збільшити довжини сторін трикутника Т3 на вказану величину. Серед трикутників Т1, Т2, Т3 визначити трикутник що має найбільшу площу.

**Математична модель:**

Для імплементації трикутника використаємо клас з назвою Triangle що міститиме відповідно методи: друкування трикутника, змінення розміру трикутника на певну сталу величину, перевантажені оператори ++,+-,+=,-= ; а також конструктори з різними типами параметрів. Використовуючи принцип дизайну архітектури декомпозиція можемо розбити трикутник на складові частини а саме вершини та сторони. Для імплементації цих сутностей використаємо класи Point та Line відповідно. Для відображення зв’язку між об’єктами цих класів застосуємо агрегацію. Для знаходження площі трикутника використаємо формулу герона. Для реалізації заданих методів використаємо стандартні доступні бібліотеки та модулі.

**1)Приклад коду на C#**

**Program.cs**

using System;

using System.Collections.Generic;

namespace Lab4\_CSharp\_

{

class Program

{

static void Main(string[] args)

{

Triangle T1 = new Triangle(new Point(3.0, 4.0), new Point(4.0, 8.0), new Point(6.0, 4.0));

T1.PrintTriangle("firstTr");

T1++;

T1.PrintTriangle("firstTr after increment");

Console.WriteLine("-------------------------------------------");

Triangle T2 = new Triangle(new Line(new Point(9.0, 4.0), new Point(3.0, 6.0)),

new Line(new Point(3.0, 6.0), new Point(6.0, 4.0)),

new Line(new Point(9.0, 4.0), new Point(6.0, 4.0)));

T2.PrintTriangle("secondTr");

T2--;

T2.PrintTriangle("secondTr after dencrement");

Console.WriteLine("-------------------------------------------");

Console.Write("Enter the value(double) by which you want to increase T3: ");

double value = double.Parse(Console.ReadLine());

Triangle T3 = new Triangle(new Line(new Point(3.0, 4.0), new Point(3.0, 8.0)),

new Line(new Point(3.0, 8.0), new Point(6.0, 4.0)));

T3.PrintTriangle("thirdTr");

T3 += value;

T3.PrintTriangle("thirdTr after being increased by value");

Console.WriteLine("-------------------------------------------");

//Calculating the triangle with largest area

List<Triangle> areasTr = new List<Triangle>() { T1, T2, T3 };

(int Number, double Area) largestTriangle = (-1, -1);

for (int i = 0; i < areasTr.Count; i++)

{

if (areasTr[i].Area > largestTriangle.Area)

largestTriangle = (i, areasTr[i].Area);

}

Console.WriteLine($"T{largestTriangle.Number+1} has the largest area of {Math.Round(largestTriangle.Area, 3)}.");

}

}

}

**Triangle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4\_CSharp\_

{

class Triangle

{

public (Point A, Point B, Point C) Vertices { get; set; }

public Line[] Sides { get; set; }

public double Area { get; set; }

#region constructorOverloads

public Triangle(Point a, Point b, Point c)

{

Vertices = (a, b, c);

Sides = new[] { new Line(a, b), new Line(b, c), new Line(a, c) };

CalculateArea();

}

public Triangle(Line a, Line b, Line c)

{

Point last = a.Endings.A == b.Endings.A ? b.Endings.B : b.Endings.A;

Vertices = (a.Endings.A, a.Endings.B, last);

Sides = new[] { a, b, c };

CalculateArea();

}

public Triangle(Line a, Line b) : this(a.Endings.A, a.Endings.B, (b.Endings.A == a.Endings.A || b.Endings.A == a.Endings.B) ? b.Endings.B : b.Endings.A) { }

#endregion constructorOverloads

#region operatorOverloads

public static Triangle operator ++(Triangle tr)

{

tr.ResizeTriangle(1);

return tr;

}

public static Triangle operator --(Triangle tr)

{

tr.ResizeTriangle(-1);

return tr;

}

public static Triangle operator +(Triangle tr, double value)

{

tr.ResizeTriangle(value);

return tr;

}

public static Triangle operator -(Triangle tr, double value)

{

tr.ResizeTriangle(-value);

return tr;

}

#endregion operatorOverloads

#region methods

public void ResizeTriangle(double value, string vertix = "A")

{

if (vertix == "A")

{

List<Point> newThirdSideCoord = new List<Point>(); //contains coordinates of third side vertices after first two sides are resized

List<Line> thirdSide = new List<Line>(Sides); //after foreach cycle resized sides will be excluded from list, so it will contain only one unresized side

//in the cycle we go through all sides of triangle and if one of its ends is the vertix which is used as initial point for scaling, we resize the side

//on its opposite vertix

foreach (var side in Sides)

{

if (side.Endings.A == Vertices.A)

{

side.ResizeLine(value, "B");

newThirdSideCoord.Add(side.Endings.B);

thirdSide.Remove(side);

}

else if (side.Endings.B == Vertices.A)

{

side.ResizeLine(value, "A");

newThirdSideCoord.Add(side.Endings.A);

thirdSide.Remove(side);

}

}

thirdSide[0].Endings = (newThirdSideCoord[0], newThirdSideCoord[1]);

thirdSide[0].Length = Math.Sqrt(Math.Pow(thirdSide[0].Endings.A.Coordinates.X - thirdSide[0].Endings.B.Coordinates.X, 2)

+ Math.Pow(thirdSide[0].Endings.A.Coordinates.Y - thirdSide[0].Endings.B.Coordinates.Y, 2));

}

this.CalculateArea();

}

public void CalculateArea()

{

//Gerons formula

double p = (Sides[0].Length + Sides[1].Length + Sides[2].Length) / 2;

Area = Math.Sqrt(p \* (p - Sides[0].Length) \* (p - Sides[1].Length) \* (p - Sides[2].Length));

}

public void PrintTriangle(string triangleIdentifyer)

{

Console.WriteLine($"Triangle {triangleIdentifyer} has elements:");

foreach (var item in Sides)

{

item.PrintLine();

}

Console.WriteLine($"Area : {Math.Round(Area, 3)}");

}

#endregion methods

}

}

**Line.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4\_CSharp\_

{

class Line

{

public (Point A, Point B) Endings { get; set; }

public double Length { get; set; }

public Line(Point a, Point b)

{

Endings = (a, b);

Length = Math.Sqrt(Math.Pow(Endings.A.Coordinates.X - Endings.B.Coordinates.X, 2) + Math.Pow(Endings.A.Coordinates.Y - Endings.B.Coordinates.Y, 2));

}

public static Line operator ++(Line line)

{

line.ResizeLine(1.0);

return line;

}

public static Line operator --(Line line)

{

line.ResizeLine(-1.0);

return line;

}

public static Line operator +(Line line, double value)

{

line.ResizeLine(value);

return line;

}

public static Line operator -(Line line, double value)

{

line.ResizeLine(-value);

return line;

}

public void ResizeLine(double value, string side = "A")

{

//перевіряємо з яким кінцем відрізка проводимо операцію

if (side == "A")

{

double unitVectorY = (this.Endings.A.Coordinates.Y - this.Endings.B.Coordinates.Y) / this.Length; //розраховуємо у частину юніт вектора поточного відрізка

double unitVectorX = (this.Endings.A.Coordinates.X - this.Endings.B.Coordinates.X) / this.Length;

if (this.Length + value > 0)// якщо зміна довжини не змінить положення робочого кінця відрізка(А) відносно іншого кінця

{

this.Endings.A.Coordinates = (this.Endings.A.Coordinates.X + unitVectorX \* value, this.Endings.A.Coordinates.Y + unitVectorY \* value);

}

}

else if (side == "B")

{

double unitVectorY = (this.Endings.B.Coordinates.Y - this.Endings.A.Coordinates.Y) / this.Length; //розраховуємо у частину юніт вектора поточного відрізка

double unitVectorX = (this.Endings.B.Coordinates.X - this.Endings.A.Coordinates.X) / this.Length;

if (this.Length + value > 0)// якщо зміна довжини не змінить положення робочого кінця відрізка(А) відносно іншого кінця

{

this.Endings.B.Coordinates = (this.Endings.B.Coordinates.X + unitVectorX \* value, this.Endings.B.Coordinates.Y + unitVectorY \* value);

}

}

this.Length += value;

}

public void PrintLine()

{

Console.WriteLine($"Coordinates of line vertices are:");

Endings.A.PrintPoint();

Console.Write("\t");

Endings.B.PrintPoint();

Console.Write("\n");

Console.WriteLine($"Length : {Math.Round(this.Length, 3)} ");

}

}

}

**Point.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab4\_CSharp\_

{

class Point

{

public (double X, double Y) Coordinates { get; set; }

public Point(double x, double y)

{

Coordinates = (x, y);

}

public void PrintPoint()

{

Console.Write($" ( x: {Math.Round(Coordinates.X, 3)} y: {Math.Round(Coordinates.Y, 3)} )");

}

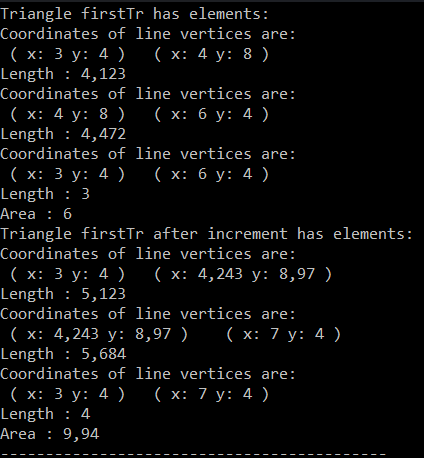
public static bool operator ==(Point a, Point b) => a.Coordinates.X == b.Coordinates.X && a.Coordinates.Y == b.Coordinates.Y;

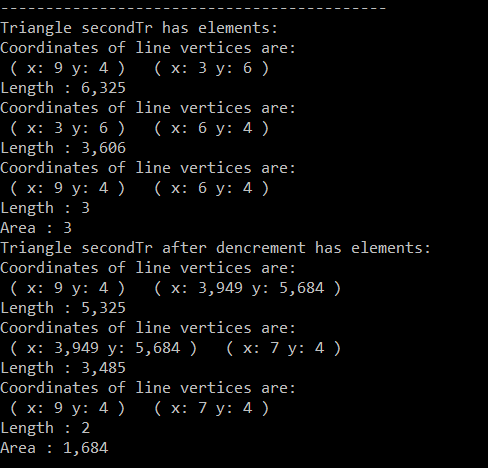
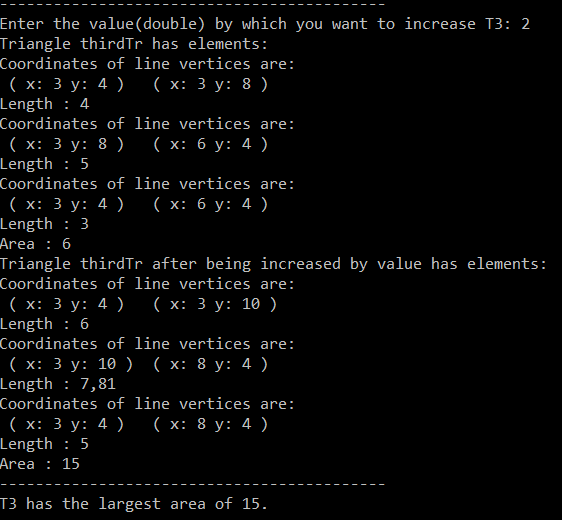
public static bool operator !=(Point a, Point b) => a.Coordinates.X != b.Coordinates.X && a.Coordinates.Y != b.Coordinates.Y;

}

}

**Виконання коду на C#:**





**Висновок:**

Отже в ході даної лабораторної роботи ми дослідили механізм перевантаження операторів в компільованій мові C# а також навчилися працювати з лб’єктами класів що містять дані оператори. Також ми вивчили механізм композиції та агрегації, декомпозиції. Дослідили функціонал бібліотеки Math та ближче познайомилися з концепцією ООП.