МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ

Національний технічний університет України

Факультет інформатики та обчислювальної техніки

«Київський політехнічний інститут імені Ігоря Сікорського»

Кафедра інформаційних систем та технологій

Звіт

з лабораторної  роботи № 1

з дисципліни «Програмування – 2.

Основи програмування»

Варіант № 5

Виконав: Валіваха Андрій

     Студент гр. ІС-11 , ФІОТ

1 курс

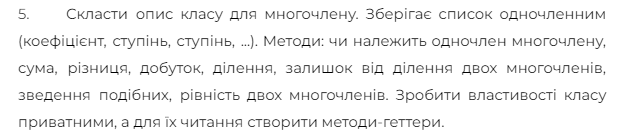
Київ 2022

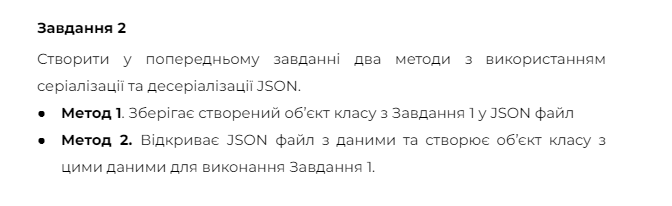
**Лабораторна робота #1**

Класи та об’єкти. Конструктори та деструктори. Модифікатори.

**Завдання.**

Створити клас с атрибутами та конструктором. У методі main() ініціалізувати створення екземплярів класу та продемонструвати роботу його методів згідно умов завдання.





using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.IO;

namespace Lab1\_OOP

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Input polynom 1 fileName:");

string fileName1 = Console.ReadLine();

string jsonString1 = File.ReadAllText(fileName1);

Polynom polynom1 = JsonSerializer.Deserialize<Polynom>(jsonString1);

Console.WriteLine("Input polynom 2 fileName:");

string fileName2 = Console.ReadLine();

string jsonString2 = File.ReadAllText(fileName2);

Polynom polynom2 = JsonSerializer.Deserialize<Polynom>(jsonString2);

Console.WriteLine("Polynom 1 initial:");

Console.WriteLine(polynom1.ConvertToString());

Console.WriteLine("Polynom 1 simpified:");

Polynom polynom1Simplified = polynom1.Simplified();

Console.WriteLine(polynom1Simplified.ConvertToString());

Console.WriteLine(" ");

Console.WriteLine("Polynom 2 initial:");

Console.WriteLine(polynom2.ConvertToString());

Console.WriteLine("Polynom 2 simpified:");

Polynom polynom2Simplified = polynom2.Simplified();

Console.WriteLine(polynom2Simplified.ConvertToString());

Console.WriteLine("Polynoms Sum (saved to polynomsSum):");

Polynom polynomSum = polynom1 + polynom2;

Console.WriteLine(polynomSum.ConvertToString());

string sumJson = JsonSerializer.Serialize(polynomSum);

File.WriteAllText("polynomsSum.json", sumJson);

Console.WriteLine("Polynoms Difference (saved to pylynomsDif):");

Polynom polynomDif = polynom1 - polynom2;

Console.WriteLine(polynomDif.ConvertToString());

string DifJson = JsonSerializer.Serialize(polynomDif);

File.WriteAllText("polynomsDif.json", DifJson);

Console.WriteLine("Polynoms Multiplication (saved to pylynomsMultipl):");

Polynom polynomMultipl = polynom1 \* polynom2;

Console.WriteLine(polynomMultipl.ConvertToString());

string MultiplJson = JsonSerializer.Serialize(polynomMultipl);

File.WriteAllText("polynomsMultipl.json", MultiplJson);

Console.WriteLine("Polynoms Division(saved to polynomsDiv):");

Polynom polynomDiv = polynom1 / polynom2;

Console.WriteLine(polynomDiv.ConvertToString());

string DivJson = JsonSerializer.Serialize(polynomDiv);

File.WriteAllText("polynomsDiv.json", DivJson);

Console.WriteLine("Polynoms Remainder (saved to polynomsRem):");

Polynom polynomRem = polynom1 % polynom2;

Console.WriteLine(polynomRem.ConvertToString());

string RemJson = JsonSerializer.Serialize(polynomRem);

File.WriteAllText("polynomsRem.json", RemJson);

Console.ReadLine();

}

}

class Single

{

private double \_koef;

private int \_power;

public double Koef

{

get { return \_koef; }

}

public int Power

{

get { return \_power; }

}

public Single (double koef, int power)

{

\_koef = koef;

\_power = power;

}

}

class Polynom

{

private List<Single> \_singles;

public List<Single> Singles

{

get { return \_singles; }

}

public Polynom (List<Single> singles)

{

\_singles = singles;

}

public string ConvertToString()

{

string result = "";

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

{

Single s = \_singles[i];

if (s.Koef >= 0)

{

result += "+";

}

if (s.Power == 0)

{

result += String.Format(" {0} ", s.Koef);

}

else

{

result += String.Format(" {0}x^{1} ", s.Koef, s.Power);

}

}

return result;

}

public void AddSingle(Single single)

{

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

{

Single s = \_singles[i];

if (s.Power == single.Power)

{

double newKoef = s.Koef + single.Koef;

if (newKoef != 0)

{

\_singles[i] = new Single(newKoef, s.Power);

} else

{

\_singles.RemoveAt(i);

}

return;

}

}

\_singles.Add(single);

}

public void MultiplySingle(Single single)

{

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

{

Single s = \_singles[i];

double newKoef = s.Koef \* single.Koef;

int newPower = s.Power + single.Power;

\_singles[i] = new Single(newKoef, newPower);

}

}

public void SubtractSingle(Single single)

{

Single s = new Single(-single.Koef, single.Power);

AddSingle(s);

}

public Polynom Simplified()

{

Polynom p = new Polynom(new List<Single>());

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

{

Single s = \_singles[i];

p.AddSingle(s);

}

return p;

}

public void SortSingles()

{

\_singles = \_singles.OrderByDescending(s => s.Power).ToList();

}

public static Polynom operator +(Polynom a, Polynom b)

{

Polynom p = new Polynom(new List<Single>());

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

{

Single s = a.Singles[i];

p.AddSingle(s);

}

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

{

Single s = b.Singles[i];

p.AddSingle(s);

}

return p;

}

public static Polynom operator -(Polynom a, Polynom b)

{

Polynom p = new Polynom(new List<Single>());

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

{

p.AddSingle(a.Singles[i]);

}

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

{

p.SubtractSingle(b.Singles[i]);

}

p.SortSingles();

return p;

}

public static Polynom operator \*(Polynom a, Polynom b)

{

Polynom p = new Polynom(new List<Single>());

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

{

Single aSingle = a.Singles[i];

for (int j = 0; j < b.Singles.Count; j++)

{

Single bSingle = b.Singles[j];

double newKoef = aSingle.Koef \* bSingle.Koef;

int newPower = aSingle.Power + bSingle.Power;

Single s = new Single(newKoef, newPower);

p.AddSingle(s);

}

}

return p;

}

public static Polynom operator /(Polynom a, Polynom b)

{

Polynom aSimplified = a.Simplified();

Polynom bSimplified = b.Simplified();

aSimplified.SortSingles();

bSimplified.SortSingles();

if (aSimplified.Singles[0].Power < bSimplified.Singles[0].Power)

{

return new Polynom(new List<Single>());

}

Polynom result = new Polynom(new List<Single>());

Polynom remainder = new Polynom(aSimplified.Singles);

while (remainder.Singles.Count > 0 && remainder.Singles[0].Power >= bSimplified.Singles[0].Power)

{

Single multiplier = new Single(remainder.Singles[0].Koef / bSimplified.Singles[0].Koef, remainder.Singles[0].Power - bSimplified.Singles[0].Power);

Polynom temp = new Polynom(new List<Single>());

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

{

Single s = new Single(bSimplified.Singles[i].Koef \* multiplier.Koef, bSimplified.Singles[i].Power + multiplier.Power);

temp.AddSingle(s);

}

remainder = remainder - temp;

remainder.SortSingles();

result.AddSingle(multiplier);

}

return result;

}

public static Polynom operator %(Polynom a, Polynom b)

{

Polynom aSimplified = a.Simplified();

Polynom bSimplified = b.Simplified();

aSimplified.SortSingles();

bSimplified.SortSingles();

if (aSimplified.Singles[0].Power < bSimplified.Singles[0].Power)

{

return new Polynom(new List<Single>());

}

Polynom remainder = new Polynom(aSimplified.Singles);

while (remainder.Singles.Count > 0 && remainder.Singles[0].Power >= bSimplified.Singles[0].Power)

{

Single multiplier = new Single(remainder.Singles[0].Koef / bSimplified.Singles[0].Koef, remainder.Singles[0].Power - bSimplified.Singles[0].Power);

Polynom temp = new Polynom(new List<Single>());

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

{

Single s = new Single(bSimplified.Singles[i].Koef \* multiplier.Koef, bSimplified.Singles[i].Power + multiplier.Power);

temp.AddSingle(s);

}

remainder = remainder - temp;

remainder.SortSingles();

}

return remainder;

}

public bool hasSingle(Single single)

{

Polynom p = this.Simplified();

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

{

Single q = p.Singles[i];

if (q.Koef == single.Koef && q.Power == single.Power)

{

return true;

}

}

return false;

}

public static bool operator ==(Polynom a, Polynom b)

{

Polynom aSimplified = a.Simplified();

Polynom bSimplified = b.Simplified();

aSimplified.SortSingles();

bSimplified.SortSingles();

if (aSimplified.Singles.Count != bSimplified.Singles.Count)

{

return false;

}

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

{

Single singleA = aSimplified.Singles[i];

Single singleB = bSimplified.Singles[i];

if(singleA.Koef != singleB.Koef || singleA.Power != singleB.Power)

{

return false;

}

}

return true;

}

public static bool operator !=(Polynom a, Polynom b)

{

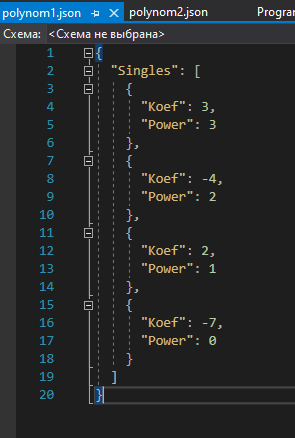
bool isEqual = a == b;

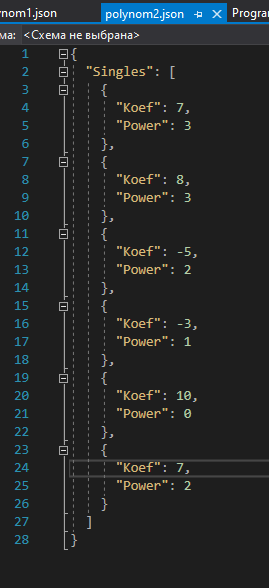
return !isEqual;

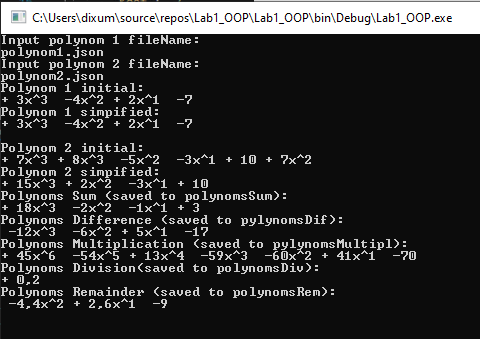
}

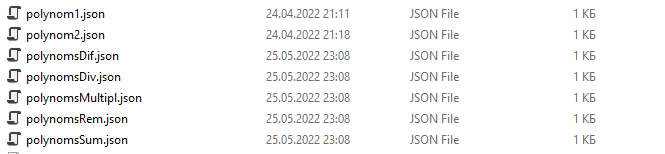
}

}









Висновок: виконавши дану лр я навчився працювати з класами і об’єктами. Використовувати конструктори, деструктори та модифікатори.