МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

ЗВІТ

з лабораторної роботи №2

з дисципліни «Програмування на C#»

Варіант 17

Виконала:

студентка групи ІТШІ-24-1

Малишко Д.І.

Перевірив:

Бібічков І.Є

Харків 2025

НАЗВА РОБОТИ: Перевантаження стандартних операцій та операцій приведення типів.

МЕТА РОБОТИ: Вивчення технології перевантаження стандартних операцій та операцій явного та неявного приведення типів для класів користувача на мові C#. Вивчення особливостей створення та використання бібліотек класів у середовищі VS.NET.

1. ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Створити проект типу ClassLibrary. Розмістити в ньому клас, що реалізує вказану в завданні сутність.

2. Включити у реалізацію класу конструктори всіх типів, функцію введення, перевизначити функцію ToString. Замість функцій доступу та ініціалізації необхідно використовувати властивості класу. Можливий виняток із цього правила у ситуаціях, коли функції доступу та ініціалізації утворюють ключову функціональність класу (у стеках, чергах, кільцях).

3. Включити до класу функції, що перевантажують операції, зазначені в завданні.

4. Для всіх варіантів включити до класу функції, що перевантажують операції true і false, дати інтерпретацію та перевантажити операції явного та неявного приведення типів.

5. Для класів, що володіють списком або динамічним масивом своїх елементів, реалізувати індексатори.

6. Також для всіх варіантів перевантажити операції “==” та “!=“ для порівняння об'єктів на рівність та на нерівність. Для сутностей, на множині яких визначено відношення порядку, перевантажити операції “>” і “=” і “<=” для порівняння на більше/менше або на більшерівно/менше-рівно.

7. Створити проект типу ConsoleApplication.

8. Підключити до окремо створеного консольного застосунку за допомогою оператору using створену бібліотеку класів. Для цього необхідно

попередньо додати до проекту посилання на файл із динамічною бібліотекою класів (.dll) (Project->Add Reference, Browse).

9. Розмістити в консольному застосунку код, що дозволяє здійснити демонстрацію використання перевантажених стандартних операцій та операцій приведення типів для реалізованої сутності.

2. ВКАЗІВКИ ДО ВИКОНАННЯ ЗАВДАННЯ

Завдання, що передбачають сутності з масивом елементів, що динамічно розміщується: обов'язково включити в клас конструктор, який використовує як один з параметрів одновимірний або двовимірний масив чисел; конструктор перетворення реалізовувати не потрібно. В цих завданнях не допускається використання стандартних контейнерів.

Варіант 17: «Поліном від однієї змінної»

Дані класу: розмірність полінома, покажчик на масив, що динамічно розміщується, коефіцієнтів при змінних.

Операції: додавання, віднімання поліномів, додавання та множення полінома на число, обчислення значення полінома для заданого значення змінної (індексатор з double).

3. ВИКОНАННЯ

Інтерфейс IPolynomial.cs

```
using System;

namespace PolynomialLibrary
{
    Ссылок: 54
    public interface IPolynomial
    {
        Ссылок: 9 | 7/7 passing
        int Degree { get; }

        Ссылок: 27 | 16/16 passing
        double this[int power] { get; set; }

        Ссылок: 1
        double this[double x] { get; }
```

```
        Ссылок: 3 | 1/1 passing
        Polynomial Add(Polynomial other);

        Ссылок: 3 | 1/1 passing
        Polynomial Sub(Polynomial other);

        Ссылок: 3 | 1/1 passing
        Polynomial Add(double number);

        Ссылок: 4 | 1/1 passing
        Polynomial Multiply(double number);

        Ссылок: 2 | 1/1 passing
        double[] ToArray();

        Ссылок: 1
        string ToString();
    }
}
```

Реалізація Polynomial.cs

```
using System;
using System.Collections.Generic;

namespace PolynomialLibrary
{
    Ссылок: 99+
    public partial class Polynomial : IPolynomial, IEquatable<Polynomial>
    {
        private int degree;
        private double[] coefficients;

        Ссылок: 9 | 7/7 passing
        int IPolynomial.Degree => degree;

        Ссылок: 27 | 16/16 passing
        double IPolynomial.this[int power]
        {
            get
            {
                if (power < 0 || power > degree) return 0;
                return coefficients[power];
            }
            set
            {
                if (power < 0)
                    throw new ArgumentOutOfRangeException(nameof(power));

                if (power > degree)
                {
                    Array.Resize(ref coefficients, power + 1);
                    degree = power;
                }

                coefficients[power] = value;
                while (degree > 0 && coefficients[degree] == 0)
                    degree--;
            }
        }
```

Ссылок: 1
```
double IPolynomial.this[double x]
{
    get
    {
        double sum = 0;
        for (int i = 0; i <= degree; i++)
            sum += coefficients[i] * Math.Pow(x, i);
        return sum;
    }
}
```

Ссылок: 3 | 3/3 passing
```
public Polynomial()
{
    degree = 0;
    coefficients = new double[1] { 0 };
}
```

Ссылок: 2 | 2/2 passing
```
public Polynomial(int degree)
{
    if (degree < 0) throw new ArgumentException("Degree cannot be negative");
    this.degree = degree;
    coefficients = new double[degree + 1];
}
```

Ссылок: 26 | 16/16 passing
```
public Polynomial(double[] arr)
{
    if (arr == null || arr.Length == 0)
        throw new ArgumentException("Array is empty");

    coefficients = new double[arr.Length];
    Array.Copy(arr, coefficients, arr.Length);

    degree = arr.Length - 1;
    while (degree > 0 && coefficients[degree] == 0)
        degree--;
}
```

Ссылок: 1 | 1/1 passing
```
public Polynomial(Polynomial p)
{
    degree = p.degree;
    coefficients = new double[p.coefficients.Length];
    Array.Copy(p.coefficients, coefficients, p.coefficients.Length);
}
```

Ссылок: 3 | 1/1 passing
```
Polynomial IPolynomial.Add(Polynomial other)
{
    int maxDeg = Math.Max(degree, other.degree);
    double[] result = new double[maxDeg + 1];

    for (int i = 0; i <= maxDeg; i++)
    {
        double a = (i <= degree) ? coefficients[i] : 0;
        double b = (i <= other.degree) ? other.coefficients[i] : 0;
        result[i] = a + b;
    }

    return new Polynomial(result);
}
```

Ссылок: 3 | 1/1 passing
```
Polynomial IPolynomial.Add(double number)
{
    double[] result = new double[coefficients.Length];
    Array.Copy(coefficients, result, coefficients.Length);
    result[0] += number;
    return new Polynomial(result);
}
```

Ссылок: 4 | 1/1 passing
```
Polynomial IPolynomial.Multiply(double number)
{
    double[] result = new double[degree + 1];
    for (int i = 0; i <= degree; i++)
        result[i] = coefficients[i] * number;

    return new Polynomial(result);
}
```

```
Ссылок: 2 | 1/1 passing
double[] IPolynomial.ToArray()
{
    double[] result = new double[coefficients.Length];
    Array.Copy(coefficients, result, coefficients.Length);
    return result;
}

Ссылок: 1
string IPolynomial.ToString() => ToString();

Ссылок: 1 | 1/1 passing
public static Polynomial operator +(Polynomial a, Polynomial b)
    => ((IPolynomial)a).Add(b);

Ссылок: 1 | 1/1 passing
public static Polynomial operator -(Polynomial a, Polynomial b)
    => ((IPolynomial)a).Sub(b);

Ссылок: 0
public static Polynomial operator +(Polynomial p, double number)
    => ((IPolynomial)p).Add(number);

Ссылок: 1 | 1/1 passing
public static Polynomial operator *(Polynomial p, double number)
    => ((IPolynomial)p).Multiply(number);

Ссылок: 0
public static Polynomial operator *(double number, Polynomial p)
    => ((IPolynomial)p).Multiply(number);

Ссылок: 2 | 1/1 passing
public static bool operator ==(Polynomial a, Polynomial b)
{
    if (ReferenceEquals(a, b)) return true;
    if (a is null || b is null) return false;
    return a.Equals(b);
}

Ссылок: 0
public static bool operator !=(Polynomial a, Polynomial b)
    => !(a == b);

Ссылок: 0
public static bool operator true(Polynomial p)
    => !(p.degree == 0 && p.coefficients[0] == 0);

Ссылок: 0
public static bool operator false(Polynomial p)
    => (p.degree == 0 && p.coefficients[0] == 0);

public static implicit operator Polynomial(double number)
    => new Polynomial(new double[] { number });

public static explicit operator double[](Polynomial p)
{
    double[] result = new double[p.coefficients.Length];
    Array.Copy(p.coefficients, result, p.coefficients.Length);
    return result;
}
```

```
        public override string ToString()
        {
            List<string> parts = new List<string>();
            for (int i = degree; i >= 0; i--)
            {
                if (coefficients[i] == 0) continue;
                string part = coefficients[i].ToString();
                if (i == 1) part += "x";
                else if (i > 1) part += $"x^{i}";
                parts.Add(part);
            }

            if (parts.Count == 0) return "0";
            return string.Join(" + ", parts);
        }

        Ссылок: 0
        public override bool Equals(object obj)
        {
            if (obj is Polynomial other)
                return Equals(other);

            return false;
        }

        Ссылок: 0
        bool IEquatable<Polynomial>.Equals(Polynomial other)
        {
            return Equals(other);
        }
```

```
        Ссылок: 0
        bool IEquatable<Polynomial>.Equals(Polynomial other)
        {
            return Equals(other);
        }

        Ссылок: 3
        public bool Equals(Polynomial other)
        {
            if (other is null) return false;
            if (degree != other.degree) return false;

            for (int i = 0; i <= degree; i++)
            {
                if (coefficients[i] != other.coefficients[i])
                    return false;
            }

            return true;
        }
        Ссылок: 0
        public override int GetHashCode()
        {
            int hashCode = 2113002308;
            hashCode = hashCode * -1521134295 + degree.GetHashCode();
            foreach (double coef in coefficients)
            {
                hashCode = hashCode * -1521134295 + coef.GetHashCode();
            }
            return hashCode;
        }
    }
}
```

Модульні тести PolynomialTest.cs

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using PolynomialLibrary;
using System;

namespace PolynomialTests
{
    [TestClass]
    Ссылок: 0
    public class PolynomialTest
    {

        [TestMethod]
        Ссылок: 0
        public void Constructor_DefaultConstructor_CreatesZeroPolynomial()
        {
            // Arrange

            // Act
            Polynomial p = new Polynomial();

            // Assert
            Assert.AreEqual(0.0, ((IPolynomial)p)[0]);
            Assert.AreEqual(0, ((IPolynomial)p).Degree);
        }


        [TestMethod]
        Ссылок: 0
        public void Constructor_DegreeConstructor_CreatesPolynomialWithZeros()
        {
            // Arrange
            int expectedDegree = 4;

            // Act
            Polynomial p = new Polynomial(expectedDegree);

            // Assert
            Assert.AreEqual(expectedDegree, ((IPolynomial)p).Degree);
            Assert.AreEqual(0, ((IPolynomial)p)[2]);
        }
        [TestMethod]
        Ссылок: 0
        public void Constructor_NegativeDegree_ThrowsArgumentException()
        {
            // Arrange

            // Act

            // Assert
            Assert.ThrowsException<ArgumentException>(() => new Polynomial(-5));
        }
```

```
[TestMethod]
Ссылок: 0
public void Constructor_ArrayConstructor_CorrectDegreeSet()
{
    // Arrange
    double[] arr = { 1, 2, 3 };

    // Act
    Polynomial p = new Polynomial(arr);

    // Assert
    Assert.AreEqual(2, ((IPolynomial)p).Degree);
}


[TestMethod]
Ссылок: 0
public void Constructor_ArrayWithLeadingZeros_CorrectDegreeReduced()
{
    // Arrange
    double[] arr = { 1, 2, 0, 0 };

    // Act
    Polynomial p = new Polynomial(arr);

    // Assert
    Assert.AreEqual(1, ((IPolynomial)p).Degree);
}

[TestMethod]
Ссылок: 0
public void Constructor_CopyConstructor_CreatesExactCopy()
{
    // Arrange
    Polynomial p1 = new Polynomial(new double[] { 2, 5, 3 });

    // Act
    Polynomial p2 = new Polynomial(p1);

    // Assert
    Assert.AreEqual(((IPolynomial)p1)[2], ((IPolynomial)p2)[2]);
    Assert.AreEqual(((IPolynomial)p1).Degree, ((IPolynomial)p2).Degree);
}
```

```
[TestMethod]
Ссылок: 0
public void Indexer_GetCoefficient_ReturnsCorrectValue()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 1, 4, 9 });

    // Act
    double value = ((IPolynomial)p)[2];

    // Assert
    Assert.AreEqual(9, value);
}
```

```
[TestMethod]
Ссылок: 0
public void Indexer_SetCoefficientOutOfRange_ExpandsDegree()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 1 });

    // Act
    ((IPolynomial)p)[3] = 5;

    // Assert
    Assert.AreEqual(5, ((IPolynomial)p)[3]);
    Assert.AreEqual(3, ((IPolynomial)p).Degree);
}


[TestMethod]
Ссылок: 0
public void Indexer_SetNegativePower_ThrowsArgumentOutOfRange()
{
    // Arrange
    Polynomial p = new Polynomial();

    // Act

    // Assert
    Assert.ThrowsException<ArgumentOutOfRangeException>(() => ((IPolynomial)p)[-1] = 10)
}

[TestMethod]
Ссылок: 0
public void Indexer_EvaluateAtPoint_ReturnsCorrectValue()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 1, 2, 3 });

    // Act
    double result = ((IPolynomial)p)[2];

    // Assert
    Assert.AreEqual(3, result);
}


[TestMethod]
Ссылок: 0
public void Add_AddTwoPolynomials_ReturnsCorrectSum()
{
    // Arrange
    Polynomial a = new Polynomial(new double[] { 1, 2 });
    Polynomial b = new Polynomial(new double[] { 3, 4 });

    // Act
    Polynomial result = ((IPolynomial)a).Add(b);

    // Assert
    Assert.AreEqual(4, ((IPolynomial)result)[0]);
    Assert.AreEqual(6, ((IPolynomial)result)[1]);
}
```

```
[TestMethod]
Ссылок: 0
public void Sub_SubtractTwoPolynomials_ReturnsCorrectDifference()
{
    // Arrange
    Polynomial a = new Polynomial(new double[] { 5, 5 });
    Polynomial b = new Polynomial(new double[] { 2, 3 });

    // Act
    Polynomial result = ((IPolynomial)a).Sub(b);

    // Assert
    Assert.AreEqual(3, ((IPolynomial)result)[0]);
    Assert.AreEqual(2, ((IPolynomial)result)[1]);
}


[TestMethod]
Ссылок: 0
public void Add_AddNumber_AddsToConstantTerm()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 2, 4 });

    // Act
    Polynomial result = ((IPolynomial)p).Add(3);

    // Assert
    Assert.AreEqual(5, ((IPolynomial)result)[0]);
    Assert.AreEqual(4, ((IPolynomial)result)[1]);
}
```

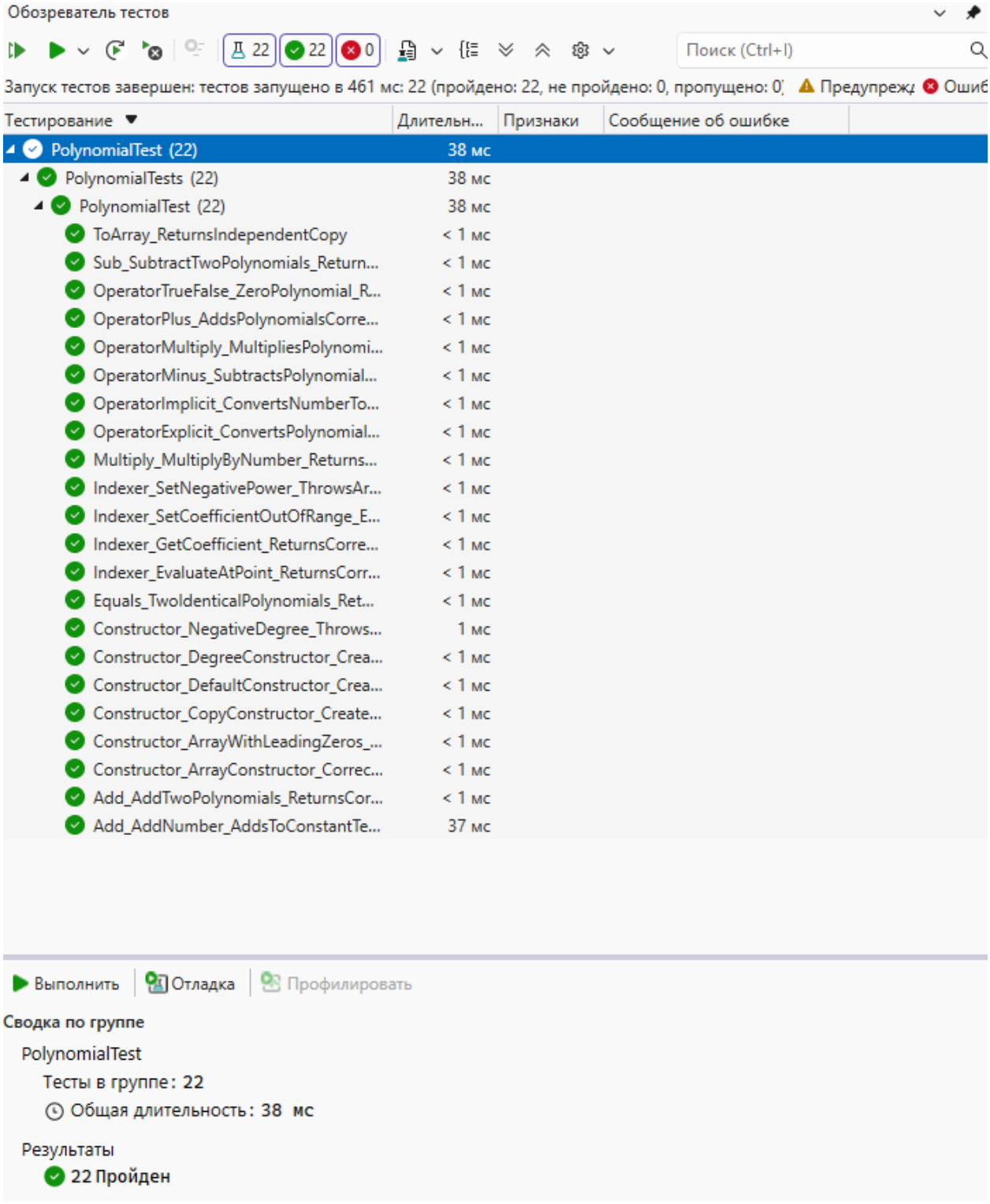
```
[TestMethod]
Ссылок: 0
public void Multiply_MultiplyByNumber_ReturnsScaledPolynomial()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 1, 2, 3 });

    // Act
    Polynomial result = ((IPolynomial)p).Multiply(2);

    // Assert
    Assert.AreEqual(2, ((IPolynomial)result)[0]);
    Assert.AreEqual(4, ((IPolynomial)result)[1]);
    Assert.AreEqual(6, ((IPolynomial)result)[2]);
}
```

```
[TestMethod]
Ссылок: 0
public void ToArray_ReturnsIndependentCopy()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 1, 2 });

    // Act
    double[] arr = ((IPolynomial)p).ToArray();
    arr[0] = 99;

    // Assert
    Assert.AreEqual(1, ((IPolynomial)p)[0]);
}


[TestMethod]
Ссылок: 0
public void OperatorPlus_AddsPolynomialsCorrectly()
{
    // Arrange
    Polynomial a = new Polynomial(new double[] { 1, 1 });
    Polynomial b = new Polynomial(new double[] { 2, 3 });

    // Act
    Polynomial c = a + b;

    // Assert
    Assert.AreEqual(3, ((IPolynomial)c)[0]);
    Assert.AreEqual(4, ((IPolynomial)c)[1]);
}
```

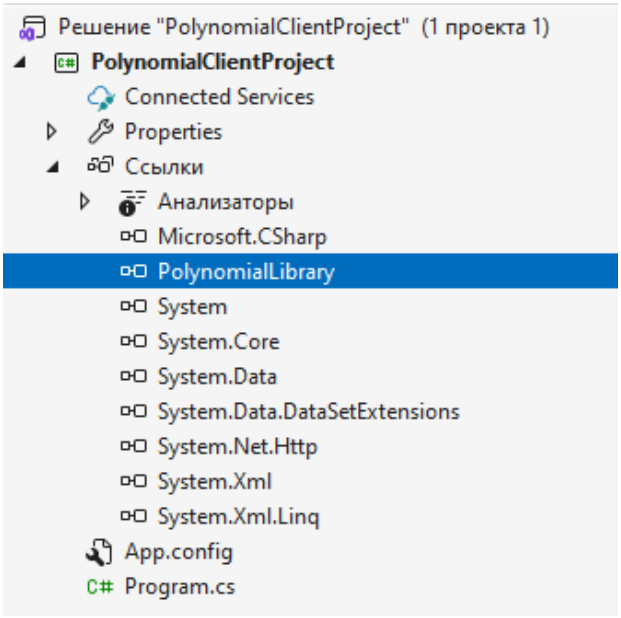
```
[TestMethod]
Ссылок: 0
public void OperatorMinus_SubtractsPolynomialsCorrectly()
{
    // Arrange
    Polynomial a = new Polynomial(new double[] { 5, 4 });
    Polynomial b = new Polynomial(new double[] { 1, 1 });

    // Act
    Polynomial c = a - b;

    // Assert
    Assert.AreEqual(4, ((IPolynomial)c)[0]);
    Assert.AreEqual(3, ((IPolynomial)c)[1]);
}


[TestMethod]
Ссылок: 0
public void OperatorMultiply_MultipliesPolynomialByNumber()
{
    // Arrange
    Polynomial p = new Polynomial(new double[] { 3, 3 });

    // Act
    Polynomial r = p * 2;

    // Assert
    Assert.AreEqual(6, ((IPolynomial)r)[0]);
    Assert.AreEqual(6, ((IPolynomial)r)[1]);
}
```

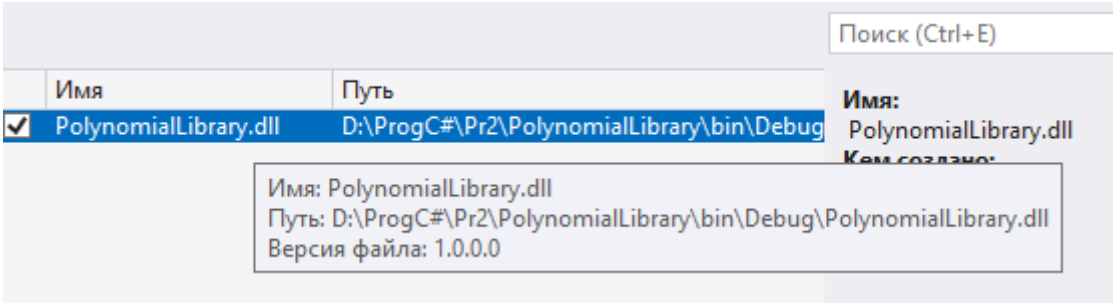
```
        [TestMethod]
        Ссылок: 0
        public void OperatorImplicit_ConvertsNumberToPolynomial()
        {
            // Arrange
            double number = 7;

            // Act
            Polynomial p = number;

            // Assert
            Assert.AreEqual(7, ((IPolynomial)p)[0]);
            Assert.AreEqual(0, ((IPolynomial)p).Degree);
        }


        [TestMethod]
        Ссылок: 0
        public void OperatorExplicit_ConvertsPolynomialToArray()
        {
            // Arrange
            Polynomial p = new Polynomial(new double[] { 4, 5 });

            // Act
            double[] arr = (double[])p;

            // Assert
            Assert.AreEqual(4, arr[0]);
            Assert.AreEqual(5, arr[1]);
        }

        [TestMethod]
        Ссылок: 0
        public void Equals_TwoIdenticalPolynomials_ReturnsTrue()
        {
            // Arrange
            Polynomial a = new Polynomial(new double[] { 1, 2 });
            Polynomial b = new Polynomial(new double[] { 1, 2 });

            // Act
            bool equals = a == b;

            // Assert
            Assert.IsTrue(equals);
        }
        [TestMethod]
        Ссылок: 0
        public void OperatorTrueFalse_ZeroPolynomial_ReturnsFalse()
        {
            // Arrange
            Polynomial p = new Polynomial();

            // Act
            bool result = p ? true : false;

            // Assert
            Assert.IsFalse(result);
        }
    }
}
```

Результати тестів



Консольний застосунок

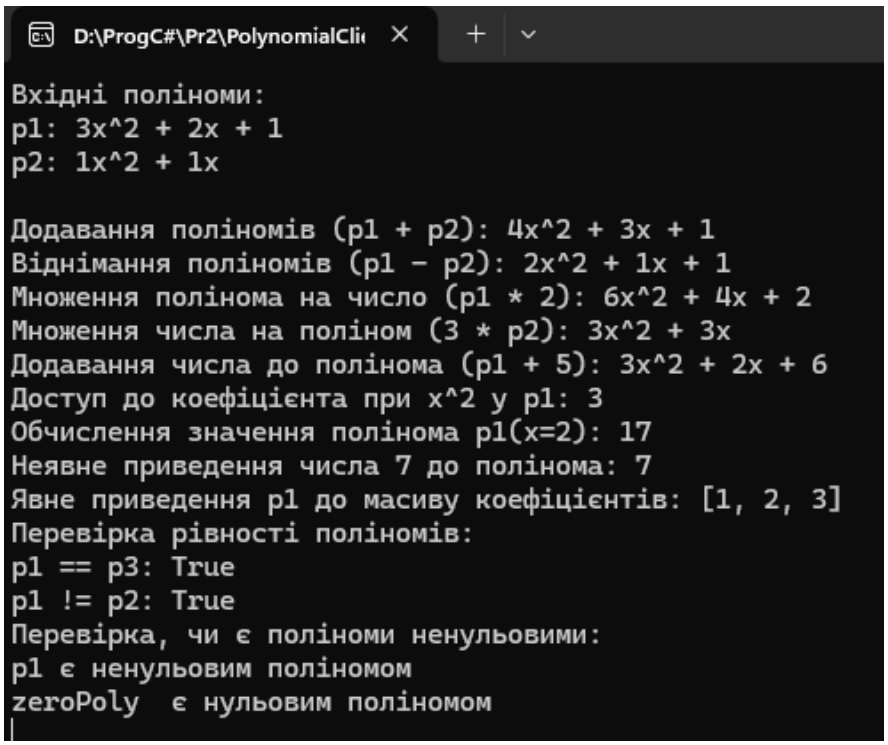





```
using PolynomialLibrary;
using System;

namespace PolynomialClientProject
{
    Ссылок: 0
    internal class Program
    {
        Ссылок: 0
        static void Main(string[] args)
        {
            Console.OutputEncoding = System.Text.Encoding.UTF8;
            Console.InputEncoding = System.Text.Encoding.UTF8;

            // Створення поліномів
            Polynomial p1 = new Polynomial(new double[] { 1, 2, 3 }); // 3x^2 + 2x + 1
            Polynomial p2 = new Polynomial(new double[] { 0, 1, 1 }); // x^2 + x

            Console.WriteLine("Вхідні поліноми:");
            Console.WriteLine("p1: " + p1);
            Console.WriteLine("p2: " + p2);
            Console.WriteLine();

            // Додавання поліномів
            Polynomial sum = p1 + p2;
            Console.WriteLine("Додавання поліномів (p1 + p2): " + sum);

            // Віднімання поліномів
            Polynomial diff = p1 - p2;
            Console.WriteLine("Віднімання поліномів (p1 - p2): " + diff);
```

```
// Множення полінома на число
Polynomial prod1 = p1 * 2;
Polynomial prod2 = 3 * p2;
Console.WriteLine("Множення полінома на число (p1 * 2): " + prod1);
Console.WriteLine("Множення числа на поліном (3 * p2): " + prod2);

// Додавання числа до полінома
Polynomial addNumber = p1 + 5;
Console.WriteLine("Додавання числа до полінома (p1 + 5): " + addNumber);

// Доступ до коефіцієнтів через індексатор
double coeffX2 = ((IPolynomial)p1)[2];
Console.WriteLine("Доступ до коефіцієнта при x^2 у p1: " + coeffX2);

// Обчислення значення полінома у точці
double valueAt2 = ((IPolynomial)p1)[2.0]; // Значення p1 при x=2
Console.WriteLine("Обчислення значення полінома p1(x=2): " + valueAt2);

// Неявне приведення числа до полінома
Polynomial pFromDouble = 7; // implicit operator
Console.WriteLine("Неявне приведення числа 7 до полінома: " + pFromDouble);

// Явне приведення полінома до масиву коефіцієнтів
double[] coeffs = (double[])p1;
Console.WriteLine("Явне приведення p1 до масиву коефіцієнтів: [" + string.Join(", ", coeffs) + "]");

// Перевірка рівності поліномів
Polynomial p3 = new Polynomial(new double[] { 1, 2, 3 });
Console.WriteLine("Перевірка рівності поліномів:");
Console.WriteLine("p1 == p3: " + (p1 == p3));
Console.WriteLine("p1 != p2: " + (p1 != p2));
```

```
            // Перевірка оператора true/false
            Polynomial zeroPoly = new Polynomial();
            Console.WriteLine("Перевірка, чи є поліноми ненульовими:");
            if (p1)
                Console.WriteLine("p1 є ненульовим поліномом");

            if (zeroPoly)
                Console.WriteLine("Нульовий полігон є ненульовим поліномом");
            else
                Console.WriteLine("Нульовий полігон є нульовим поліномом");

            Console.ReadLine();
        }
    }
}
```



Висновки

У ході виконання лабораторної роботи було реалізовано клас «Поліном від однієї змінної» відповідно до вимог варіанта. Робота була розділена на два проєкти: бібліотеку класів (ClassLibrary) та консольний застосунок (ConsoleApplication) для демонстрації роботи перевантажених операторів і механізмів приведення типів.

Важливою частиною роботи було перевантаження операторів true і false, що дає можливість визначати, чи є поліном нульовим. Крім того, реалізовано операції явного та неявного приведення типів: неявне перетворення числа до полінома та явне перетворення полінома до масиву коефіцієнтів.

Виконання даної лабораторної роботи дозволило детально ознайомитися з механізмами перевантаження операторів у C#, правилами організації класів у бібліотеці класів та підключенням таких бібліотек до інших проєктів. Реалізований функціонал продемонстрував важливість правильної роботи з динамічними масивами, а також показав, що перевантаження операторів значно покращує зручність використання користувацьких типів даних.