**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: **«**Операции и перегруженные методы класса**»**

Выполнил: студент гр. ИТП-21

Кислов Я.В.

Проверил: преподаватель

Карась О.В.

Гомель 2025

**Цель работы:**

Необходимо решить задачу, согласно вариант. При создании классов руководствоваться Code Convention. Весь код должен содержать элементы документирования. В реализованных классах должны быть конструкторы с параметрами и по умолчанию, а также необходимые свойства и методы. Для сравнения объектов использовать перегруженный оператор отношения. Разработать модульные тесты для верификации созданных классов. Все классы должны быть размещены в библиотеке классов. Модульные тесты – в отдельном проекте. В отдельном проекте реализовать консольный интерфейс.

Вариант 2

**Создать класс «Матрица», в котором описать следующие элементы:**

**- закрытое поле – матрица целых чисел;**

**- закрытое строковое поле для хранения имени матрицы;**

**- свойства для определения количества строк и столбцов массива;**

**- индексатор для доступа к элементам поля-массива;**

**- конструктор с параметрами;**

**- методы ввода и вывода матрицы;**

**- перегруженные методы для вычисления произведения отрицательных элементов матрицы с выводом результата (параметр – имя матрицы) и без вывода;**

**- операция поэлементного сложения матриц одинаковой размерности;**

**- операции `true` и `false` (массив истинный, если в нем есть ненулевые элементы).**

**Разработать программу, выполняющую следующие действия:**

**- Ввод и вывод трех матриц;**

**- Вычисление и вывод произведения отрицательных элементов в каждом массиве;**

**- Вычисление суммы массивов с одинаковыми размерностями;**

**- Если произведение отрицательных элементов первого массива больше заданного числа, а в третьей матрице есть ненулевые элементы, увеличить все отрицательные элементы этого массива на значение минимального среди элементов последней строки третьего массива.**

**Ход работы:**

Создаем класс Matrix, в котором выполняем все условия задания. В моем случае это сложение через перегрузку оператора “+” матрицы на матрицу, а также. У нас есть закрытое поле с двумерным массивом, это и будет наша матрица, также создаем конструктор, индексатор и 2 статического перегруженных операторов с словом operator для реализации операций с матрицами. Также были сделаны модульные тесты в MSTest. На рисунке 1 показано выполнение модульных тестов. Код программы расположен в приложении А.

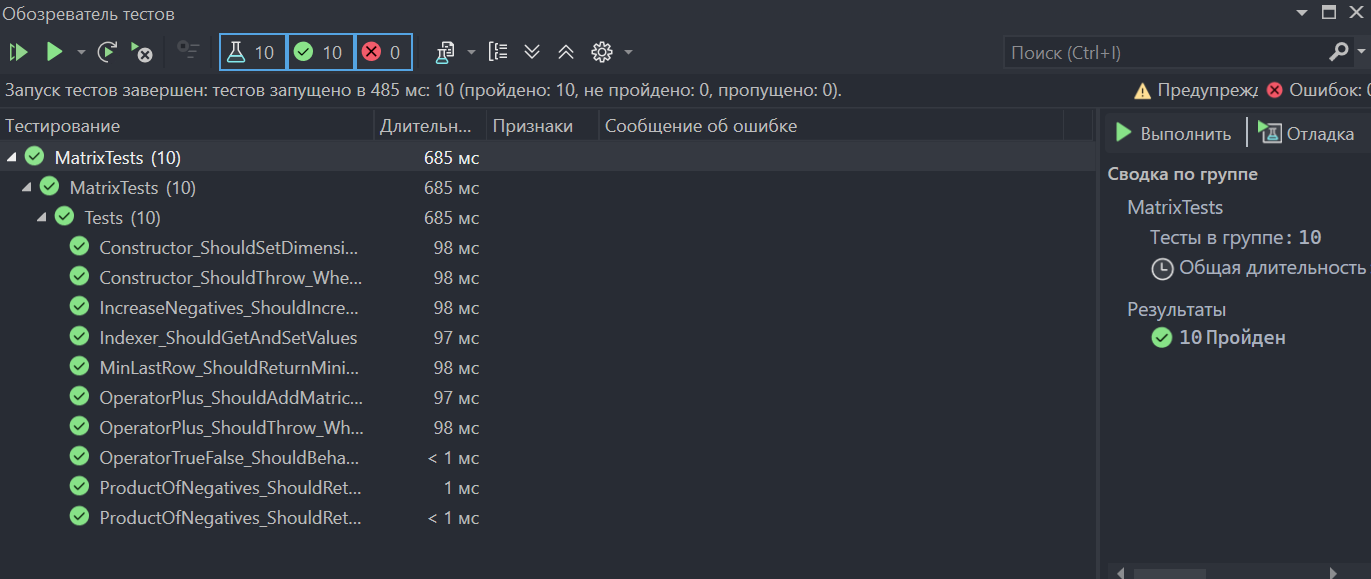


Рисунок 1 – Результат выполнения тестов

**Вывод:** в ходе работы был успешно разработан класс для работы с матрицами, реализующий операцию сложения матриц. Библиотека классов была протестирована с помощью модульных тестов и интегрирована в консольное приложение. Результаты верификации подтвердили корректность выполнения операций.

**Приложение А**

**(обязательно)**

Код программы

namespace MatrixLibrary

{

/// <summary>

/// Класс, представляющий матрицу и операции над ней.

/// </summary>

public class Matrix

{

private int[,] \_matrix;

private string \_nameMatrix;

/// <summary>

/// Количество строк в матрице.

/// </summary>

public int CountRows { get; }

/// <summary>

/// Количество столбцов в матрице.

/// </summary>

public int CountColumns { get; }

/// <summary>

/// Создает новый экземпляр класса <see cref="Matrix"/>.

/// </summary>

/// <param name="matrix">Двумерный массив значений матрицы.</param>

/// <param name="nameMatrix">Имя матрицы.</param>

/// <exception cref="ArgumentNullException">Выбрасывается, если <paramref name="matrix"/> или <paramref name="nameMatrix"/> равны null.</exception>

public Matrix(int[,] matrix, string nameMatrix)

{

if (matrix is null || nameMatrix is null)

{

throw new ArgumentNullException("Matrix is null!");

}

\_matrix = matrix;

\_nameMatrix = nameMatrix;

CountRows = matrix.GetLength(0);

CountColumns = matrix.GetLength(1);

}

/// <summary>

/// Доступ к элементам матрицы по индексу.

/// </summary>

/// <param name="row">Номер строки (начиная с 0).</param>

/// <param name="col">Номер столбца (начиная с 0).</param>

/// <returns>Значение элемента матрицы.</returns>

public int this[int row, int col]

{

get => \_matrix[row, col];

set => \_matrix[row, col] = value;

}

/// <summary>

/// Ввод матрицы с консоли.

/// </summary>

/// <returns>Двумерный массив, заполненный значениями из консоли.</returns>

public int[,] InputMatrix()

{

Console.WriteLine("Введите кол-во строк и столбцов матрицы: ");

int rows = Int32.Parse(Console.ReadLine());

int columns = Int32.Parse(Console.ReadLine());

int[,] matrix = new int[rows, columns];

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

{

for (int j = 0; j < columns; j++)

{

Console.WriteLine($"Matrix[{i},{j}] = ");

matrix[i, j] = Int32.Parse(Console.ReadLine());

}

}

return matrix;

}

/// <summary>

/// Вывод матрицы на консоль.

/// </summary>

public void Print()

{

Console.WriteLine($"Matrix {\_nameMatrix}:");

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

{

for (int j = 0; j < CountColumns; j++)

Console.Write($"{\_matrix[i, j]} ");

Console.WriteLine();

}

}

/// <summary>

/// Определяет, является ли матрица "истиной" (если есть хотя бы один ненулевой элемент).

/// </summary>

/// <param name="m">Матрица для проверки.</param>

/// <returns><c>true</c>, если хотя бы один элемент матрицы не равен нулю, иначе <c>false</c>.</returns>

public static bool operator true(Matrix m)

{

foreach (int val in m.\_matrix)

if (val != 0)

return true;

return false;

}

/// <summary>

/// Определяет, является ли матрица "ложью" (все элементы равны нулю).

/// </summary>

/// <param name="m">Матрица для проверки.</param>

/// <returns><c>true</c>, если все элементы равны нулю, иначе <c>false</c>.</returns>

public static bool operator false(Matrix m)

{

foreach (int val in m.\_matrix)

if (val != 0)

return false;

return true;

}

/// <summary>

/// Операция сложения двух матриц одинакового размера.

/// </summary>

/// <param name="a">Первая матрица.</param>

/// <param name="b">Вторая матрица.</param>

/// <returns>Новая матрица, являющаяся результатом сложения.</returns>

/// <exception cref="ArgumentException">Выбрасывается, если размеры матриц не совпадают.</exception>

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

{

if (a.CountRows != b.CountRows || a.CountColumns != b.CountColumns)

throw new ArgumentException("Матрицы разной размерности!");

int[,] result = new int[a.CountRows, a.CountColumns];

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

for (int j = 0; j < a.CountColumns; j++)

result[i, j] = a[i, j] + b[i, j];

return new Matrix(result, a.\_nameMatrix + "+" + b.\_nameMatrix);

}

/// <summary>

/// Вычисляет произведение всех отрицательных элементов матрицы.

/// </summary>

/// <returns>Произведение отрицательных чисел, либо 0, если их нет.</returns>

public long MulOfNegatives()

{

long product = 1;

bool found = false;

foreach (int val in \_matrix)

{

if (val < 0)

{

product \*= val;

found = true;

}

}

if (found)

return product;

return 0;

}

/// <summary>

/// Вычисляет произведение всех отрицательных элементов матрицы с выводом результата.

/// </summary>

/// <param name="matrixName">Имя матрицы для отображения в консоли.</param>

/// <returns>Произведение отрицательных чисел, либо 0, если их нет.</returns>

public long MulOfNegatives(string matrixName)

{

long result = MulOfNegatives();

Console.WriteLine($"Произведение отрицательных элементов матрицы {matrixName}: {result}");

return result;

}

/// <summary>

/// Увеличивает все отрицательные элементы матрицы на указанное значение.

/// </summary>

/// <param name="addValue">Значение, на которое увеличиваются отрицательные элементы.</param>

public void IncreaseNegatives(int addValue)

{

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

for (int j = 0; j < CountColumns; j++)

if (\_matrix[i, j] < 0)

\_matrix[i, j] += addValue;

}

/// <summary>

/// Возвращает минимальный элемент последней строки матрицы.

/// </summary>

/// <returns>Минимальное значение в последней строке.</returns>

public int MinLastRow()

{

int lastRow = CountRows - 1;

int min = \_matrix[lastRow, 0];

for (int j = 1; j < CountColumns; j++)

if (\_matrix[lastRow, j] < min)

min = \_matrix[lastRow, j];

return min;

}

}

}

Приложение А1 – Код класса Matrix

using MatrixLibrary;

namespace MatrixApplication

{

internal class Program

{

static void Main(string[] args)

{

Console.Write("Введите имя матрицы: ");

string name1 = Console.ReadLine();

Matrix A = new Matrix(CreateMatrix(), name1);

Console.Write("Введите имя матрицы: ");

string name2 = Console.ReadLine();

Matrix B = new Matrix(CreateMatrix(), name2);

Console.Write("Введите имя матрицы: ");

string name3 = Console.ReadLine();

Matrix C = new Matrix(CreateMatrix(), name3);

A.Print();

B.Print();

C.Print();

A.MulOfNegatives(name1);

B.MulOfNegatives(name2);

C.MulOfNegatives(name3);

Matrix D = A + B;

Console.WriteLine("Сумма А+B: ");

D.Print();

long prodA = A.MulOfNegatives();

Console.WriteLine("Задайте число: ");

double number = Double.Parse(Console.ReadLine());

if (prodA > number && C.MulOfNegatives() != 0)

{

int minLastRow = C.MinLastRow();

A.IncreaseNegatives(minLastRow);

Console.WriteLine("\nA после изменения:");

A.Print();

}

Console.ReadKey();

}

public static int[,] CreateMatrix()

{

Console.Write("Введите количество строк: ");

int rows = int.Parse(Console.ReadLine());

Console.Write("Введите количество столбцов: ");

int cols = int.Parse(Console.ReadLine());

int[,] values = new int[rows, cols];

Console.WriteLine("Введите элементы матрицы построчно:");

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

{

for (int j = 0; j < cols; j++)

{

Console.Write($"[{i},{j}] = ");

values[i, j] = int.Parse(Console.ReadLine());

}

}

return values;

}

}

}

Приложение А.2 – Код главного класса Program

using MatrixLibrary;

namespace MatrixTests

{

[TestClass]

public sealed class Tests

{

[TestMethod]

[ExpectedException(typeof(ArgumentNullException))]

public void Constructor\_ShouldThrow\_WhenNullPassed()

{

var m = new Matrix(null, null);

}

[TestMethod]

public void Constructor\_ShouldSetDimensionsCorrectly()

{

int[,] arr = { { 1, 2 }, { 3, 4 } };

var m = new Matrix(arr, "A");

Assert.AreEqual(2, m.CountRows);

Assert.AreEqual(2, m.CountColumns);

}

[TestMethod]

public void Indexer\_ShouldGetAndSetValues()

{

int[,] arr = { { 1, 2 }, { 3, 4 } };

var m = new Matrix(arr, "A");

Assert.AreEqual(3, m[1, 0]);

m[1, 0] = 99;

Assert.AreEqual(99, m[1, 0]);

}

[TestMethod]

public void OperatorPlus\_ShouldAddMatrices()

{

int[,] aArr = { { 1, 2 }, { 3, 4 } };

int[,] bArr = { { 5, 6 }, { 7, 8 } };

var a = new Matrix(aArr, "A");

var b = new Matrix(bArr, "B");

var result = a + b;

Assert.AreEqual(6, result[0, 0]);

Assert.AreEqual(12, result[1, 1]);

}

[TestMethod]

[ExpectedException(typeof(ArgumentException))]

public void OperatorPlus\_ShouldThrow\_WhenDifferentDimensions()

{

var a = new Matrix(new int[,] { { 1, 2 } }, "A");

var b = new Matrix(new int[,] { { 3, 4 }, { 5, 6 } }, "B");

var res = a + b;

}

[TestMethod]

public void ProductOfNegatives\_ShouldReturnCorrectProduct()

{

int[,] arr = { { -1, 2 }, { -3, 4 } };

var m = new Matrix(arr, "A");

long product = m.MulOfNegatives();

Assert.AreEqual(3, product);

}

[TestMethod]

public void ProductOfNegatives\_ShouldReturnZero\_WhenNoNegatives()

{

int[,] arr = { { 1, 2 }, { 3, 4 } };

var m = new Matrix(arr, "A");

Assert.AreEqual(0, m.MulOfNegatives());

}

[TestMethod]

public void IncreaseNegatives\_ShouldIncreaseOnlyNegatives()

{

int[,] arr = { { -5, 2 }, { -3, -1 } };

var m = new Matrix(arr, "A");

m.IncreaseNegatives(2);

Assert.AreEqual(-3, m[0, 0]);

Assert.AreEqual(2, m[0, 1]);

Assert.AreEqual(-1, m[1, 0]);

Assert.AreEqual(1, m[1, 1]);

}

[TestMethod]

public void MinLastRow\_ShouldReturnMinimumValueInLastRow()

{

int[,] arr = { { 5, 6 }, { 7, 2 }, { 9, -1 } };

var m = new Matrix(arr, "A");

Assert.AreEqual(-1, m.MinLastRow());

}

[TestMethod]

public void OperatorTrueFalse\_ShouldBehaveCorrectly()

{

var nonZero = new Matrix(new int[,] { { 1, 0 } }, "A");

var zero = new Matrix(new int[,] { { 0, 0 } }, "B");

if (nonZero)

{

Assert.IsTrue(true);

}

else

{

Assert.Fail("Expected nonZero matrix to evaluate true");

}

if (zero)

{

Assert.Fail("Expected zero matrix to evaluate false");

}

else

{

Assert.IsTrue(true);

}

}

}

}

Приложение А.3 – Код MSTest