**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 программирование»

на тему: «Операции и перегруженные методы

класса»

Выполнил: студент гр. ИТП-21

Король В.Н.

Принял: преподаватель-стажер

Гуменников Е.Д.

Гомель 2022

**Цель работы:** изучить основы языка программирования *C*#. Научится создавать классы их основные возможности. Изучить такие понятия как поля класса, конструктор класса и методы класса. Изучить методы перегрузки на языке программирования *C*#. Научится создавать внутри классов операторы. Изучить работу модульных тестов в *Visual Studio*.

**Задание**

1. Необходимо решить задачу, согласно варианта (таблица 3)

2. При создании классов руководствоваться Code Convention

3. Весь код должен содержать элементы документирования

4. В реализованных классах должны быть конструкторы с параметрами и по

умолчанию, а также необходимые свойства и методы

5. Для сравнения объектов использовать перегруженный оператор

отношения

6. Разработать модульные тесты для верификации созданных классов

7. Все классы должны быть размещены в библиотеке классов

8. Модульные тесты – в отдельном проекте

9. В отдельном проекте реализовать консольный интерфейс.

**Вариант 14**

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

• закрытое поле – матрица целых чисел,

• свойства для определения количества строк и столбцов матрицы,

• индексатор для доступа к элементам поля-массива,

• конструктор с параметрами,

• методы ввода и вывода матрицы,

• перегруженные методы для вычисления суммы квадратов положительных

элементов матрицы, расположенных ниже минимального среди элементов строк

с номерами кратными n (параметр – n>=0), и вычисления суммы квадратов

отрицательных элементов матрицы (без параметров),

• операция поэлементного вычитания матриц одинаковой размерности,

• операции ≤ и ≥ (матрица X≤Y, если сумма квадратов положительных элементов

матрицы Х меньше или равна сумме квадратов положительных элементов

матрицы Y).

Разработать программу, выполняющую следующие действия:

− Ввод и вывод трех матриц А, В и С;

− Вычисление и вывод суммы квадратов отрицательных элементов каждой

матрицы;

− Вычисление А-В и B-A-C, если это возможно;

− Если A≤ B≤ C, заменить все отрицательные элементы матриц А и B на значение

суммы квадратов положительных элементов расположенных ниже минимального

среди элементов строк с номерами кратными 3 в матрице С.

**Ход Работы**

По условию задания был создан класс *Matrix* содержащий *private* поле где хранится сама матрица, основные методы и операторы для работы с матрицей.

После объявления класса был создан конструктор с перегрузкой. Если в конструктор не передавать параметры то по умолчанию создается пустая матрица, в другом случае матрица будет равняться переданному параметром двумерному массиву.

Далее были созданы методы *CreateMatrix* который позволяют задать матрицу параметрами в этот метод передаются два целых числа, где первое число это число строк, а второе это число столбцов. Пример работы метода *CreateMatrix* указан на рисунке 1.

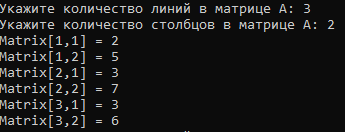


Рисунок 1 - Пример работы метода *CreateMatrix*

Метод *ShowMatrix* позволяет вывести матрицу в консоль. Пример работы метода *ShowMatrix* указан на рисунке 1.



Рисунок 2 - Пример работы метода *ShowMatrix*

Для получения количества строк и столбцов были созданы методы *GetNumberOfColumns* и *GetNumberOfLines* которые при помощи метода *GetLength*() выводят количество элементов во вложенных массива.

Для получения значения элемента по индексу был создан метод *GetIndexOfMatrix* в который передается два параметра (номер строки и номер столбца) и возвращает значение нужного элемента. Если такого элемента нет то возвращается значение -1.

Для нахождения суммы квадратов всех элементов меньших нуля был создан перегруженный метод *GetSumSquerad*, который при помощи оператора *foreach* перебирает все элементы массива. Если у объекта класса вызвать этот метод и параметром передать туда значение *int* то *GetSumSquerad* найдет сумма квадратов положительных элементов в тех строках которые располагаются ниже минимального элемента матрицы и номера которых кратны переданному параметром числу.

Для вычитания матриц был создан перегруженный оператор - который параметрам принимает два объекта класса *Matrix.* Если переданные матрицы нельзя вычесть то оператор возвращает пустую матрицу. В противном случае возвращается результат вычислений.

Для сравнения матриц были созданы перегруженные операторы >= и <= которые принимают два объекта класса *Matrix* и возвращают значение типа *bool* сравнение матриц производится по принципу суммы квадратов отрицательных элементов, для нахождения данной суммы был создан метод *GetSumSquaredForOperator.*

После выполнения задания в основном файле было созданное меню в соответствии с условием задания. Пример меню указан на рисунке 3.

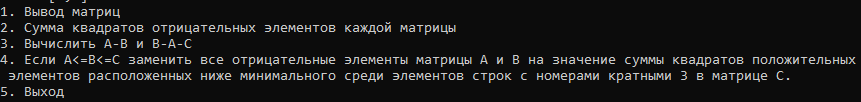


Рисунок 3 - Пример работы меню

После создания класса и меню программы были созданы модульные тесты для основных методов и операторов. Все тесты указаны на в листинге отчета. Результат их работы указан на рисунке 4.

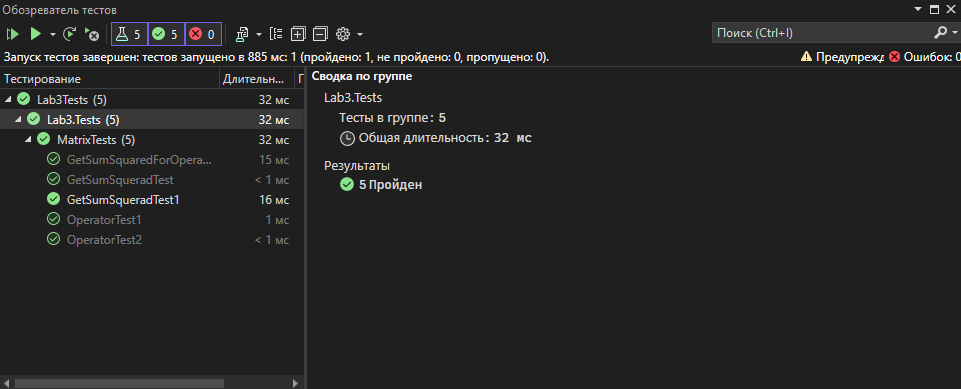


Рисунок 4 - Результат работу модульных тестов

**Вывод:** в ходе выполнения лабораторной работы были методы перегрузки на языка программирования *C*#. Научился создавать операторы. Научился делать модульные тесты в *Visual Studio*. Был создан класс *Matrix* в котором была использована информация полученная при выполнении предыдущих работ.

**Листинг класса *Matrix***

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public class Matrix

{

private int[,] \_matrix;

public Matrix(int[,] matrix)

{

\_matrix = matrix;

}

public Matrix()

{

\_matrix = new int[,]{};

}

public void CreateMatrix(int lines, int columns)

{

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

for (int i = 0; i < matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix.GetLength(1); j++)

{

Console.Write($"Matrix[{i+1},{j+1}] = ");

matrix[i, j] = Convert.ToInt32(Console.ReadLine());

}

}

\_matrix = matrix;

}

public void ShowMatrix()

{

for (int i = 0; i < \_matrix.GetLength(0); i++)

{

for (int j = 0; j < \_matrix.GetLength(1); j++)

{

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

}

Console.WriteLine();

}

}

public int GetNumberOfColumns()

{

return \_matrix.GetLength(1);

}

public int GetNumberOfLines()

{

return \_matrix.GetLength(0);

}

public int GetIndexOfMatrix(int x, int y)

{

if(x >= 0 && x < \_matrix.GetLength(0) && y >= 0 && y < \_matrix.GetLength(1))

{

return \_matrix[x, y];

}

return -1;

}

public int GetSumSquaredForOperator()

{

int sum = 0;

foreach(var elem in \_matrix)

{

if (elem > 0)

{

sum += (int)Math.Pow(elem, 2);

}

}

return sum;

}

public int GetSumSquerad()

{

int sum = 0;

foreach (var elem in \_matrix)

{

if (elem < 0)

{

sum += (int)Math.Pow(elem, 2);

}

}

return sum;

}

public int GetSumSquerad(int n)

{

int sum = 0;

int min = \_matrix[0, 0];

int minLine = 0;

for (int i = 0; i < \_matrix.GetLength(0); i++)

{

for (int j = 0; j < \_matrix.GetLength(1); j++)

{

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

{

min = \_matrix[i, j];

minLine = i;

}

}

}

if(minLine + 1 <= \_matrix.GetLength(0))

{

for (int i = minLine + 1; i < \_matrix.GetLength(0); i++)

{

if ((i+1) % n == 0)

{

for (int j = 0; j < \_matrix.GetLength(1); j++)

{

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

{

sum += (int)Math.Pow(\_matrix[i, j], 2);

}

}

}

}

}

return sum;

}

public void ChangeElements(Matrix matrix)

{

for (int i = 0; i < \_matrix.GetLength(0); i++)

{

for(int j = 0; j < \_matrix.GetLength(1); j++)

{

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

{

\_matrix[i, j] = matrix.GetSumSquerad(3);

}

}

}

}

public static Matrix operator -(Matrix matrix1, Matrix matrix2)

{

Matrix result = new Matrix();

if (matrix1.\_matrix.GetLength(0) == matrix2.\_matrix.GetLength(0) && matrix1.\_matrix.GetLength(1) == matrix2.\_matrix.GetLength(1))

{

result.\_matrix = new int[matrix1.\_matrix.GetLength(0), matrix1.\_matrix.GetLength(1)];

for (int i = 0; i < matrix1.\_matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix1.\_matrix.GetLength(1); j++)

{

result.\_matrix[i, j] = matrix1.\_matrix[i, j] - matrix2.\_matrix[i, j];

}

}

}

return result;

}

public static bool operator <=(Matrix matrix1, Matrix matrix2)

{

return matrix1.GetSumSquaredForOperator() <= matrix2.GetSumSquaredForOperator();

}

public static bool operator >=(Matrix matrix1, Matrix matrix2)

{

return matrix1.GetSumSquaredForOperator() >= matrix2.GetSumSquaredForOperator();

}

public static bool operator ==(Matrix matrix1, Matrix matrix2)

{

if (matrix1.\_matrix.GetLength(0) == matrix2.\_matrix.GetLength(0) && matrix1.\_matrix.GetLength(1) == matrix2.\_matrix.GetLength(1))

{

for (int i = 0; i < matrix1.\_matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix1.\_matrix.GetLength(1); j++)

{

if (matrix1.\_matrix[i, j] != matrix2.\_matrix[i, j])

{

return false;

}

}

}

return true;

}

return false;

}

public static bool operator !=(Matrix matrix1, Matrix matrix2)

{

if (matrix1.\_matrix.GetLength(0) == matrix2.\_matrix.GetLength(0) && matrix1.\_matrix.GetLength(1) == matrix2.\_matrix.GetLength(1))

{

for (int i = 0; i < matrix1.\_matrix.GetLength(0); i++)

{

for (int j = 0; j < matrix1.\_matrix.GetLength(1); j++)

{

if (matrix1.\_matrix[i, j] != matrix2.\_matrix[i, j])

{

return true;

}

}

}

return false;

}

return true;

}

}

}

**Листинг класса *MatrixTests***

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Lab3;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3.Tests

{

[TestClass()]

public class MatrixTests

{

[TestMethod()]

public void GetSumSqueradTest()

{

Matrix matrix = new Matrix(new int[,] { {-1, 2}, {-2, -4} });

int expect = 21;

Assert.IsTrue(expect == matrix.GetSumSquerad());

}

[TestMethod()]

public void GetSumSqueradTest1()

{

Matrix matrix = new Matrix(new int[,] { { -10, 2 }, { 2, 4 } });

int expect = 20;

Assert.IsTrue(expect == matrix.GetSumSquerad(1));

}

[TestMethod()]

public void OperatorTest1()

{

Matrix expect = new Matrix(new int[,] { { 0, 0 }, { 0, 0 } });

Matrix matrix1 = new Matrix(new int[,] { { -10, 2 }, { 2, 4 } });

Matrix matrix2 = new Matrix(new int[,] { { -10, 2 }, { 2, 4 } });

Matrix act = new Matrix();

act = matrix2 - matrix1;

Assert.IsTrue(expect == act);

}

[TestMethod()]

public void OperatorTest2()

{

Matrix matrix1 = new Matrix(new int[,] { { 1, 1 }, { 1, 1 } });

Matrix matrix2 = new Matrix(new int[,] { { 2, 2 }, { 2,2 } });

Assert.IsTrue(matrix1 <= matrix2);

}

[TestMethod()]

public void GetSumSquaredForOperatorTest()

{

Matrix matrix2 = new Matrix(new int[,] { { 2, -2 }, { 2, -2 } });

int expect = 8;

Assert.IsTrue(expect <= matrix2.GetSumSquaredForOperator());

}

}

}