Министерство науки и высшего образованияРоссийской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический университет

имени В.Ф.Уткина»

Кафедра «Электронные вычислительные машины»

Отчет по лабораторной работе №3

на тему

«Перегрузка методов и операторов»

по дисциплине  
**«Визуальное программирование»**

Выполнили:

Студенты группы №140

Бригада 5

Сафаров Д. А.

Тимохин Е. С.

Проверили:  
ст. преп. Хизриева Н.И.

ст. преп. Бастрычкин А.С.

**Цель работы**

Изучить механизм перегрузки на языке C#.

**Задание**

1) Создать класс Matrix со свойствами Rows и Columns типа uint (неотрицательные целые, количество строк и столбцов соответственно), доступными только для чтения и с закрытым полем:**double [,] elements**;

2) Создать конструктор, который принимает в качестве аргументов размерность матрицы и затем создает матрицу заданного размера, заполненную нулями: **public Matrix(uint rows, uint columns);**

3) Создать копирующий конструктор: **public Matrix(Matrix matrix);**

4) Создать индексатор, который позволяет получить и задать значение элемента матрицы: **public double this[uint row, uint column].**В тело индексатора нужно добавить обработку исключений в случае попытки обращения по индексам, выходящим за границы матрицы;

5) Переопределить метод **ToString()**, который будет выводить количество строк, столбцов и элементов матрицы;

6) Перегрузить метод **ToString()**, который будет выводить в строку n первых элементов матрицы;

7) Перегрузить **операторы +, -, \* ;**

8) Реализовать в классе Matrix **интерфейс ICloneable**.

9) Реализовать в классе Matrix **интерфейс IComparable**.

**Практическая часть**

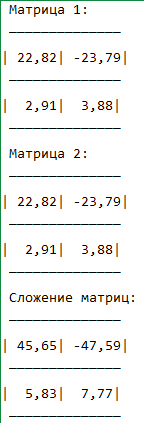
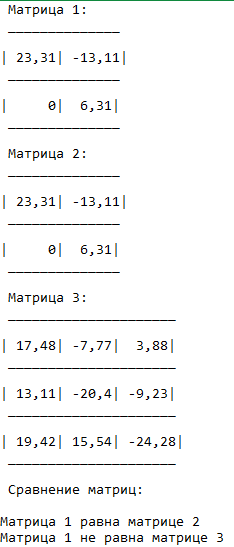
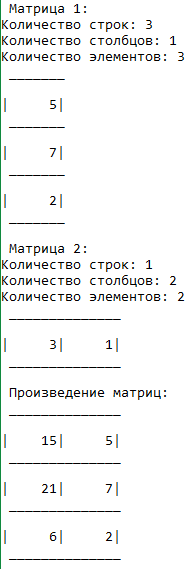
Код программы представлен в приложении А. Результат её работы представлен на рисунке 1.

Рисунок 1 – Примеры работы программы

Приложение А. Текст программы

**Matrix.cs**

using System.Text;

namespace MatrixLib

{

public class Matrix : ICloneable, IComparable

{

private double[,] \_elements; // Массив значений

public uint Rows { get; private set; } // Количество строк

public uint Columns { get; private set; } // Количество столбцов

// Конструктор с задаваемыми размерами

public Matrix(uint rows, uint columns)

{

this.Rows = rows;

this.Columns = columns;

\_elements = new double[Rows, Columns];

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

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

\_elements[i, j] = 0;

}

// Конструктор копирования

public Matrix(Matrix matrix)

{

this.Rows = matrix.Rows;

this.Columns = matrix.Columns;

\_elements = new double[Rows, Columns];

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

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

\_elements[i, j] = matrix.\_elements[i, j];

}

// Индексатор доступа к элементам

public double this[uint row, uint column]

{

get

{

if (row < 0 || row >= Rows) throw new Exception("get:

'row' argument out of range!");

if (column < 0 || column >= Columns) throw new

Exception("get: 'column' argument out of range!");

return \_elements[row, column];

}

set

{

if (row < 0 || row >= Rows) throw new Exception("set:

'row' argument out of range!");

if (column < 0 || column >= Columns) throw new

Exception("set: 'column' argument out of range!");

\_elements[row, column] = value;

}

}

// Возврат строки с количеством строк, столбцов и элементов

public override string ToString()

{

return

$"Количество строк: {Rows}" +

$"\nКоличество столбцов: {Columns}" +

$"\nКоличество элементов: {Rows\*Columns}";

}

// Возврат n первых элементов в одной строке

public string ToString(uint n)

{

StringBuilder nValuesStringBuilder = new StringBuilder();

uint i = 0;

foreach (var value in \_elements)

{

nValuesStringBuilder.Append(value);

nValuesStringBuilder.Append(' ');

if (++i >= n) break;

}

return nValuesStringBuilder.ToString();

}

// Возвращает результат сложения двух матриц

public static Matrix operator +(Matrix left, Matrix right)

{

if (left.Rows != right.Rows || left.Columns != right.Columns) throw new Exception("operator+: matrices must have equal

dimensions!");

Matrix result = new Matrix(left);

for (uint i = 0; i < result.Rows; i++)

for (uint j = 0; j < result.Rows; j++)

result[i, j]+= right[i, j];

return result;

}

// Возвращает результат вычитания правой матрицы из левой

public static Matrix operator -(Matrix left, Matrix right)

{

if (left.Rows != right.Rows || left.Columns != right.Columns) throw new Exception("operator-: matrices must have equal

dimensions!");

Matrix result = new Matrix(left);

for (uint i = 0; i < result.Rows; i++)

for (uint j = 0; j < result.Rows; j++)

result[i, j]-= right[i, j];

return result;

}

// Возвращает результат произведения левой матрицы на правую

public static Matrix operator \*(Matrix left, Matrix right)

{

if (left.Columns != right.Rows) throw new

Exception("operator\*: left matrix columns number must be equal to right matrix rows number!");

Matrix result = new Matrix(left.Rows, right.Columns);

for (uint i = 0; i < (result.Rows); i++)

for (uint j = 0; j < (result.Columns); j++)

{

double c = 0;

for (uint k = 0; k < right.Rows; k++)

c += left[i, k] \* right[k, j];

result[i, j] = c;

}

return result;

}

// Реализация метода Clone интерфейса ICloneable

public object Clone()

{

return new Matrix(this);

}

// Реализация метода CompareTo интерфейса ICompareable

// Возвращает:

// 0 – Матрицы равны или это один объект

// 1 – Матрицы равных размеров, сумма вызывающей матрицы больше

// -1 – Матрицы равных размеров, сумма вызывающей матрицы меньше

// 2 – Вызывающая матрица больше по размерам

// -2 – Вызывающая матрица меньше по размерам

public int CompareTo(object? obj)

{

if (obj == null) throw new Exception("CompareTo: object is

null!");

if (obj == this) return 0;

Matrix matrixObject = obj as Matrix;

if (this.Rows != matrixObject.Rows || this.Columns != matrixObject.Columns)

return (this.Rows \* this.Columns > matrixObject.Rows \* matrixObject.Columns) ? 2 : -2;

bool unequalSum = false;

for (uint i = 0; i < Rows && !unequalSum; i++)

for (uint j = 0; j < Columns && !unequalSum; j++)

if (this.\_elements[i, j] != matrixObject.\_elements[i, j]) unequalSum = true;

if (unequalSum)

{

double sumMain = 0, sumObj = 0;

foreach (var element in this.\_elements)

sumMain += element;

foreach (var element in matrixObject.\_elements)

sumObj += element;

return (sumMain > sumObj) ? 1 : -1;

}

return 0;

}

// Вывод матрицы в консоль

public void Print()

{

Console.Write(' ');

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

for (int i = 0; i < 7; i++)

Console.Write("\_");

Console.WriteLine();

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

{

Console.WriteLine();

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

{

Console.Write("| ");

Console.Write("{0:F2}",

Math.Round(\_elements[i,j],2).ToString().PadLeft(5));

}

Console.WriteLine("|");

Console.Write(' ');

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

for (int k = 0; k < 7; k++)

Console.Write("\_");

Console.WriteLine();

}

}

// Заполнение матрицы случайными числами

public void Rand()

{

Random rnd = new Random();

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

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

\_elements[i, j] = Math.Round(rnd.Next(0, 51) \* Math.Pow(-1, rnd.Next(1, 3)) \* 0.4856, 3);

}

}

}