

**Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»**

Кафедра интеллектуальных информационных технологий

**Отчет по лабораторной работе №1
по курсу «Проектирование программ в интеллектуальных системах»**

Выполнил студент группы 921703

Кравцов М.С.

Проверил:

Садовский М.Е.

МИНСК 2020

Тема:

Объектно-ориентированное программирование.

Цель работы:

Изучить базовые принципы объектно-ориентированного программирования.

Задание:

Реализовать на языке C++ один из нижеперечисленных вариантов и написать и сгенерировать документацию при помощи `doxygen`. Для возможности тестирования классов написать тестовую программу с меню или набор unit-тестов. В случае написания unit-тестов необходимо проверить не менее 30 тестов случаев с использованием библиотеки `UnitTest++`.

Каждый из реализованных классов должен иметь следующие свойства:

- инкапсуляция;
- отделение консольного пользовательского интерфейса программы от реализации класса;
- конструктор копирования и оператор `=`, если это уместно;
- деструктор, если это необходимо;
- операторы для сравнения (операторы `==`, `!=`);
- операторы для работы с потоками ввода (`>>`) и вывода (`<<`);
- взаимная независимость класса и пользовательского интерфейса, использующего его;
- отделение объявления класса в `h`-файл, а реализации — в `cpp`-файл.

1.7.2 Вещественная матрица (усложненный)

Класс должен реализовывать следующие дополнительные возможности:

- сложение двух матриц (операторы `+`, `+=`);
- сложение матрицы с числом (операторы `+`, `+=`);
- вычитание двух матриц (операторы `-`, `-=`);
- вычитание из матрицы числа (операторы `-`, `-=`);
- произведение двух матриц (оператор `*`);
- произведение матрицы на число (операторы `*`, `*=`);
- деление матрицы на число (операторы `/`, `/=`);
- возведение матрицы в степень (оператор `^`, `^=`);
- вычисление детерминанта;
- вычисление нормы;

Выполнение заданий:

Заголовочный файл класса Matrix (Matrix.h)

```
#pragma once
#ifndef _MATRIX_
#define _MATRIX_

#include <vector>
#include <iostream>
#include <cmath>

using namespace std;

/**
 * \file
 * \brief Matrix
 * In mathematics, a matrix is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns.
 */

class Matrix
{
public:
    Matrix();

    /// \brief sizeX - Matrix rows, sizeY - Matrix cols
    Matrix(const int& sizeX, const int& sizeY);

    ///\brief Constructor to initialize a matrix
    Matrix(const Matrix& other);

    ///\brief Constructor that creates a matrix based on a vector
    ///\param other - vector
    Matrix(const vector<vector<double>>& other);

    ///\brief Matrix comparison operator
    ///\param other - matrix
    bool operator==(const Matrix& other) const;

    ///\brief Matrix comparison operator
    ///\param other - matrix
    bool operator!=(const Matrix& other) const;

    ///\brief Matrix assignment operator
    ///\param other - matrix
    Matrix& operator=(const Matrix& other);

    ///\brief Addition operator of a matrix and a number
    ///\param other - number
    Matrix operator+(const double& other);

    ///\brief Addition operator with matrix and number assignment
    ///\param other - number
    Matrix& operator+=(const double& other);
```

```

///\brief Operator for adding two matrices
///\param other - matrix
Matrix operator+(const Matrix& other);

///\brief Addition operator with assignment of two matrices
///\param other - matrix
Matrix& operator+=(const Matrix& other);

///\brief Difference operator of a matrix and a number
///\param other - number
Matrix operator-(const double& other);

///\brief The operator of the difference between the assignment matrix and the number
///\param other - number
Matrix& operator--(const double& other);

///\brief Difference operator of two matrices
///\param other - matrix
Matrix operator-(const Matrix& other);

///\brief The difference operator of two matrices with assignment
///\param other - matrix
Matrix& operator--(const Matrix& other);

///\brief Matrix multiplication operator with a number
///\param other - number
Matrix operator*(const double& other);

///\brief Matrix and number multiplication operator with assignment
///\param other - number
Matrix& operator*=(const double& other);

///\brief Multiplication operator for two matrices
///\param other - matrix
Matrix operator*(const Matrix& other);

///\brief Operator for dividing a matrix by a number
///\param other - number
Matrix operator/(const double& other);

///\brief Operator for dividing a matrix by a number with assignment
///\param other - number
Matrix& operator/=(const double& other);

///\brief The operator matrix of degree
///\param other - number
Matrix operator^(const int& other);

///\brief Matrix exponentiation operator with assignment
///\param other - number
Matrix& operator^=(const int& other);

///\brief Calculation of the matrix determinant
///\return If method squareMatrixCheck() is true - return determinant \n else - return 0
double getDeterminantOfMatrix();

///\brief Calling the normOne() method for a matrix
double getNormOneOfMatrix() const;

///\brief Calling the normTwo() method for a matrix
double getNormTwoOfMatrix() const;

///\brief Calling the normThree() method for a matrix
double getNormThreeOfMatrix() const;

///\brief Checking the matrix for compatibility
///\return if the matrices are compatible - returns true \n else - returns false
bool matrixCompatibilityCheck(const Matrix& other) const;

///\brief Checking the possibility of matrix multiplication
///\return if the number of columns in the first matrix is equal to the number of rows in the second matrix - returns true
/// \n else - returns false
bool matrixMultiplicationCheck(const Matrix& other) const;

///\brief Checks the matrix for squaring
///\return if the matrix is square - returns true \n else - returns false
bool squareMatrixCheck() const;

///\brief Output operator
///\return returns stream
friend ostream& operator<<(ostream& os, const Matrix& matrix);

///\brief Input operator
///\return returns stream
friend istream& operator>>(istream& stream, Matrix& matrix);

```

```

private:
    ///\brief A two-dimensional vector that stores the values of the matrix
    vector<vector<double>> arr;

    ///\brief Calculating the minor of a matrix
    ///\return matrix - matrix
    Matrix minor(Matrix matrix, const int& i, const int& j);

    ///\brief Calculating the determinant of a matrix
    ///\return matrix - matrix
    double determinant(const Matrix& matrix);

    ///\brief Finding the first norm of the matrix
    double normOne() const;

    ///\brief Finding the second norm of the matrix
    double normTwo() const;

    ///\brief Finding the third norm of the matrix
    double normThree() const;

};

#endif _MATRIX_

```

Переменные класса:

- 1) arr – двумерный вектор, который хранит значение матрицы

Конструкторы:

```

Matrix::Matrix() //конструктор, который создает объект, но не заполняет матрицу
{
}

Matrix::Matrix(const int& sizeX, const int& sizeY) //конструктор, который заполняет матрицу X на Y нулями
{
    arr.resize(sizeX);
    for (int i = 0; i < arr.size(); i++)
    {
        arr[i].resize(sizeY);
        for (int k = 0; k < arr[i].size(); k++)
        {
            arr[i][k] = 0;
        }
    }
}

Matrix::Matrix(const Matrix& other) //конструктор копирования
{
    arr.resize(other.arr.size());
    for (int i = 0; i < arr.size(); i++)
    {
        arr[i].resize(other.arr[i].size());
        for (int k = 0; k < arr[i].size(); k++)
        {
            arr[i][k] = other.arr[i][k];
        }
    }
}

Matrix::Matrix(const vector<vector<double>>& other) //конструктор, который заполняет матрицу числами из двумерного вектора
{
    arr.resize(other.size());
    for (int i = 0; i < arr.size(); i++)
    {
        arr[i].resize(other[i].size());
        for (int k = 0; k < arr[i].size(); k++)
        {
            arr[i][k] = other[i][k];
        }
    }
}

```

Перегрузка операторов:

```
bool Matrix::operator==(const Matrix& other) const //перегрузка оператора сравнения (==)
{
    if (matrixCompatibilityCheck(other)) //сравнение матриц на соответствие размеров (вызов функции сравнения размеров матриц)
    {
        for (int i = 0; i < this->arr.size(); i++)
        {
            for (int k = 0; k < this->arr[i].size(); k++)
            {
                if (this->arr[i][k] != other.arr[i][k])
                {
                    return false;
                }
            }
        }
        return true;
    }
    else
    {
        return false;
    }
}

bool Matrix::operator!=(const Matrix& other) const //перегрузка оператора сравнения (!=)
{
    return !(this == other); //вызов оператора сравнения (==) и инверсия полученного значения
}

Matrix& Matrix::operator=(const Matrix& other) //перегрузка оператора присваивания
{
    arr.clear();
    arr.resize(other.arr.size());
    for (int i = 0; i < arr.size(); i++)
    {
        arr[i].resize(other.arr[i].size());
        for (int k = 0; k < arr[i].size(); k++)
        {
            arr[i][k] = other.arr[i][k];
        }
    }
    return *this;
}

Matrix Matrix::operator+(const double& other) //перегрузка оператора сложения матрицы и числа
{
    Matrix temp(this->arr.size(), this->arr[0].size());
    for (int i = 0; i < temp.arr.size(); i++)
    {
        for (int k = 0; k < temp.arr[i].size(); k++)
        {
            temp.arr[i][k] = this->arr[i][k] + other;
        }
    }
    return temp;
}

Matrix& Matrix::operator+=(const double& other) //перегрузка оператора сложения-присваивания матрицы и числа
{
    return *this = *this + other; //сложение матрицы и числа и присваивание полученного значения исходной матрице
}

Matrix Matrix::operator+(const Matrix& other) //перегрузка оператора сложения двух матриц
{
    if (this->matrixCompatibilityCheck(other)) //вызов функции сравнения матриц на соответствие размеров (если размеры не совпадают, то результатом будет первое слагаемое)
    {
        Matrix temp(this->arr.size(), this->arr[0].size());
        for (int i = 0; i < temp.arr.size(); i++)
        {
            for (int k = 0; k < temp.arr[i].size(); k++)
            {
                temp.arr[i][k] = this->arr[i][k] + other.arr[i][k];
            }
        }
        return temp;
    }
    else
    {
        return *this;
    }
}

Matrix& Matrix::operator+=(const Matrix& other) //перегрузка оператора сложения-присваивания двух матриц
{
    return *this = *this + other;
}

Matrix Matrix::operator-(const double& other) //перегрузка оператора вычитания числа от матрицы
{
    Matrix temp(this->arr.size(), this->arr[0].size());
    for (int i = 0; i < temp.arr.size(); i++)
    {
        for (int k = 0; k < temp.arr[i].size(); k++)
        {
            temp.arr[i][k] = this->arr[i][k] - other;
        }
    }
    return temp;
}

Matrix& Matrix::operator-=(const double& other) //перегрузка оператора вычитания-присваивания числа от матрицы
{
    return *this = *this - other;
}
```

```

Matrix Matrix::operator-(const Matrix& other) //непергрузка оператора вычитания матрицы от матрицы
{
    if (this->matrixCompatibilityCheck(other)) //вызов функции сравнения матриц на соответствие размеров (если размеры не совпадают, то результатом будет первое слагаемое)
    {
        Matrix temp(this->arr.size(), this->arr[this->arr.size() - 1].size());
        for (int i = 0; i < temp.arr.size(); i++)
        {
            for (int k = 0; k < temp.arr[i].size(); k++)
            {
                temp.arr[i][k] = this->arr[i][k] - other.arr[i][k];
            }
        }
        return temp;
    }
    else
    {
        return *this;
    }
}

Matrix& Matrix::operator-=(const Matrix& other) //непергрузка оператора вычитания-присваивания матрицы от матрицы
{
    return *this = *this - other;
}

Matrix Matrix::operator*(const double& other) //непергрузка оператора умножения матрицы на число
{
    Matrix temp(this->arr.size(), this->arr[this->arr.size() - 1].size());
    for (int i = 0; i < temp.arr.size(); i++)
    {
        for (int k = 0; k < temp.arr[i].size(); k++)
        {
            temp.arr[i][k] = this->arr[i][k] * other;
        }
    }
    return temp;
}

Matrix& Matrix::operator*=(const double& other) //непергрузка оператора умножения-присваивания матрицы на число
{
    return *this = *this * other;
}

Matrix Matrix::operator*(const Matrix& other) //непергрузка оператора умножения двух матриц
{
    if (this->matrixMultiplicationCheck(other)) //вызов функции, которая проверяет можно ли умножить матрицу *this на матрицу other (в противном случае перегрузка возвращает первый множитель)
    {
        Matrix temp(this->arr.size(), other.arr[other.arr.size() - 1].size());
        for (int i = 0; i < temp.arr.size(); i++)
        {
            for (int k = 0; k < temp.arr[i].size(); k++)
            {
                for (int j = 0; j < other.arr.size(); j++)
                {
                    temp.arr[i][k] = temp.arr[i][k] + (this->arr[i][j] * other.arr[j][k]);
                }
            }
        }
        return temp;
    }
    else
    {
        return *this;
    }
}

Matrix Matrix::operator/(const double& other) //непергрузка оператора деления матрицы на число
{
    if (other != 0) //проверка не является ли число 0, если да, то оператор возвращает делимое (исходную матрицу)
    {
        Matrix temp(this->arr.size(), this->arr[this->arr.size() - 1].size());
        for (int i = 0; i < temp.arr.size(); i++)
        {
            for (int k = 0; k < temp.arr[i].size(); k++)
            {
                temp.arr[i][k] = this->arr[i][k] / other;
            }
        }
        return temp;
    }
    else
    {
        return *this;
    }
}

Matrix& Matrix::operator/=(const double& other) //непергрузка оператора деления-присваивания матрицы на число
{
    return *this = *this / other;
}

Matrix Matrix::operator^(const int& other) //непергрузка оператора возведения матрицы в степень
{
    if (this->squareMatrixCheck() && other > 0) //вызов функции, которая проверяет является ли матрица квадратной и проверяет степень, в которую необходимо возвести матрицу
    {
        //если матрица не квадратная, либо степень возведения < 0, то возвращается исходная матрица
        Matrix temp(*this);
        for (int i = 1; i < other; i++)
        {
            temp = temp * *this;
        }
        return temp;
    }
    else
    {
        return *this;
    }
}

Matrix& Matrix::operator^=(const int& other) //непергрузка оператора возведения в степень-присваивания для матрицы
{
    return *this = *this ^ other;
}

```

Геттеры:

```
double Matrix::getNormOneOfMatrix() const //функция, которая возвращает 1-ую норму матрицы
{
    return (this->normOne()); //вычисление первой нормы матрицы
}

double Matrix::getNormTwoOfMatrix() const //функция, которая возвращает 2-ую норму матрицы
{
    return (this->normTwo()); //вычисление второй нормы матрицы
}

double Matrix::getNormThreeOfMatrix() const //функция, которая возвращает 3-ую норму матрицы
{
    return (this->normThree()); //вычисление третьей нормы матрицы
}

double Matrix::getDeterminantOfMatrix() //функция которая возвращает детерминант матрицы
{
    if (squareMatrixCheck()) //вызов функции, которая проверяет является ли матрица квадратной (в противном случае результатом будет 0)
    {
        return determinant(*this); //функция вычисления детерминанта матрицы
    }
    else
    {
        return 0;
    }
}
```

Функция, которая определяет являются ли матрицы одного размера (необходимое условие для сложения и вычитания двух матриц):

```
bool Matrix::matrixCompatibilityCheck(const Matrix& other) const
{
    if (this->arr.size() == other.arr.size() && this->arr[this->arr.size() - 1].size() == other.arr[other.arr.size() - 1].size())
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Функция, которая определяет являются ли матрицы совместимыми для умножения:

```
bool Matrix::matrixMultiplicationCheck(const Matrix& other) const //функция, которая определяет являются ли матрицы совместимыми для умножения
{
    if (this->arr[this->arr.size() - 1].size() == other.arr.size()) //если кол-во столбцов матрицы A равно кол-ву строк матрицы B, то такие матрицы можно перемножить
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Функция, которая определяет является ли матрица квадратной (необходимое условие для возведения матрицы в степень):

```
bool Matrix::squareMatrixCheck() const //функция, которая определяет является ли матрица квадратной (необходимое условие для возведения матрицы в степень)
{
    if (this->arr.size() == this->arr[this->arr.size() - 1].size() && this->arr.size() > 1) //если число строк равно числу столбцов, то матрица квадратная
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Вычисление детерминанта матрицы


```

Matrix Matrix::minor(Matrix matrix, const int& i, const int& j) //дополнительный минор квадратной матрицы – определитель матрицы, полученной из исходной вычеркиванием i-ой строки и j-ого столбца.
{
    matrix.arr.erase(matrix.arr.begin() + i);
    for (int z = 0; z < matrix.arr.size(); z++)
    {
        matrix.arr[z].erase(matrix.arr[z].begin() + j);
    }
    return matrix;
}

double Matrix::determinant(const Matrix& matrix) //вычисление определителя матрицы NxN методом рекурсии
{
    int size = matrix.arr[0].size();
    if (size == 1) //условие выхода из рекурсии
    {
        return matrix.arr[0][0];
    }
    int signum = 1;
    int summ = 0;
    int j = 0;
    for (int i = 0; i < matrix.arr[0].size(); i++) //разложение по первой строке
    {
        summ += matrix.arr[0][i] * signum * determinant(minor(matrix, 0, j)); //детерминант матрицы NxN - равен алгебраической сумме n! членов.
        signum *= -1; //Каждый член представляет собой произведение n - элементов взятых по одному из каждой строки и каждого столбца таблицы.
        j++; //Член берется со знаком плюс, если перестановки образованные первыми(i) и вторыми индексами элементов(j) имеет одинаковую четность
        //и со минус в противном случае
    }
    return summ;
}

```

Функции вычисление нормы матрицы по 3 различным формулам:

```

double Matrix::normOne() const //функция вычисление первой нормы матрицы - максимум суммы модулей элементов в строке
{
    double max = 0, num = 0;
    for (int i = 0; i < this->arr.size(); i++)
    {
        for (int k = 0; k < this->arr[i].size(); k++)
        {
            num += abs(this->arr[i][k]);
        }
        if (num > max)
        {
            max = num;
            num = 0;
        }
        else
        {
            num = 0;
        }
    }
    return max;
}

double Matrix::normTwo() const //функция вычисления второй нормы матрицы - максимум суммы модулей элементов в столбце
{
    double max = 0, num = 0;
    for (int i = 0; i < this->arr[this->arr.size() - 1].size(); i++)
    {
        for (int k = 0; k < this->arr.size(); k++)
        {
            num += abs(this->arr[k][i]);
        }
        if (num > max)
        {
            max = num;
            num = 0;
        }
        else
        {
            num = 0;
        }
    }
    return max;
}

double Matrix::normThree() const //функция вычисления третьей нормы матрицы - квадратный корень из суммы квадратов элементов
{
    double temp = 0;
    for (int i = 0; i < this->arr.size(); i++)
    {
        for (int k = 0; k < this->arr[i].size(); k++)
        {
            temp += this->arr[i][k] * this->arr[i][k];
        }
    }
    return sqrt(temp);
}

```

Операторы ввода и вывод:

```
ostream& operator<<(ostream& stream, const Matrix& matrix)//перегрузка оператора вывода для матрицы
{
    for (int i = 0; i < matrix.arr.size(); i++)
    {
        for (int k = 0; k < matrix.arr[i].size(); k++)
        {
            stream << matrix.arr[i][k] << " ";
        }
        stream << endl;
    }
    return stream;
}

istream& operator>>(istream& stream, Matrix& matrix)//перегрузка оператора ввода для матрицы
{
    int size;
    matrix.arr.clear();
    stream >> size;
    matrix.arr.resize(size);
    stream >> size;
    for (int i = 0; i < matrix.arr.size(); i++)
    {
        matrix.arr[i].resize(size);
        for (int k = 0; k < matrix.arr[i].size(); k++)
        {
            stream >> matrix.arr[i][k];
        }
    }
    return stream;
}
```

Примеры документации, сгенерированной при помощи Doxygen:

Public Member Functions

Matrix (const int &sizeX, const int &sizeY) sizeX - Matrix rows, sizeY - Matrix cols
Matrix (const Matrix &other) Constructor to initialize a matrix.
Matrix (const vector< vector< double >> &other) Constructor that creates a matrix based on a vector. More...
bool operator== (const Matrix &other) const Matrix comparison operator. More...
bool operator!= (const Matrix &other) const Matrix comparison operator. More...
Matrix & operator= (const Matrix &other) Matrix assignment operator. More...
Matrix operator+ (const double &other) Addition operator of a matrix and a number. More...
Matrix & operator+= (const double &other) Addition operator with matrix and number assignment. More...
Matrix operator+ (const Matrix &other) Operator for adding two matrices. More...
Matrix & operator+= (const Matrix &other) Addition operator with assignment of two matrices. More...
Matrix operator- (const double &other) Difference operator of a matrix and a number. More...

Matrix & operator-= (const double &other)	The operator of the difference between the assignment matrix and the number. More...
Matrix operator- (const Matrix &other)	Difference operator of two matrices. More...
Matrix & operator-= (const Matrix &other)	The difference operator of two matrices with assignment. More...
Matrix operator* (const double &other)	Matrix multiplication operator with a number. More...
Matrix & operator*= (const double &other)	Matrix and number multiplication operator with assignment. More...
Matrix operator* (const Matrix &other)	Multiplication operator for two matrices. More...
Matrix operator/ (const double &other)	Operator for dividing a matrix by a number. More...
Matrix & operator/= (const double &other)	Operator for dividing a matrix by a number with assignment. More...
Matrix operator^ (const int &other)	The operator matrix of degree. More...
Matrix & operator^= (const int &other)	Matrix exponentiation operator with assignment. More...
double getDeterminantOfMatrix ()	Calculation of the matrix determinant. More...
double getNormOneOfMatrix () const	Calling the normOne() method for a matrix.
double getNormTwoOfMatrix () const	Calling the normTwo() method for a matrix.
double getNormThreeOfMatrix () const	Calling the normThree() method for a matrix.
bool matrixCompatibilityCheck (const Matrix &other) const	Checking the matrix for compatibility. More...
bool matrixMultiplicationCheck (const Matrix &other) const	Checking the possibility of matrix multiplication. More...
bool squareMatrixCheck () const	Checks the matrix for squaring. More...

Friends

ostream & operator<< (ostream &os, const Matrix &matrix)	Output operator. More...
istream & operator>> (istream &stream, Matrix &matrix)	Input operator. More...

Constructor & Destructor Documentation

◆ Matrix()

Matrix::Matrix (const vector< vector< double >> & other)

Constructor that creates a matrix based on a vector.

Parameters

other - vector

Member Function Documentation

◆ getDeterminantOfMatrix()

double Matrix::getDeterminantOfMatrix ()

Calculation of the matrix determinant.

Returns

If method `squareMatrixCheck()` is true - return determinant
else - return 0

◆ matrixCompatibilityCheck()

bool Matrix::matrixCompatibilityCheck (const Matrix & other) const

Checking the matrix for compatibility.

Returns

if the matrices are compatible - returns true
else - returns false

◆ matrixMultiplicationCheck()

bool Matrix::matrixMultiplicationCheck (const Matrix & other) const

Checking the possibility of matrix multiplication.

Returns

if the number of columns in the first matrix is equal to the number of rows in the second matrix - returns true
else - returns false

◆ operator!==()

bool Matrix::operator!= (const Matrix & other) const

Matrix comparison operator.

Parameters

other - matrix

◆ operator*() [1/2]

Matrix Matrix::operator* (const double & other)

Matrix multiplication operator with a number.

Parameters

other - number

◆ operator*() [2/2]

Matrix Matrix::operator* (const Matrix & other)

Multiplication operator for two matrices.

Parameters

other - matrix

◆ operator*=()

Matrix & Matrix::operator*= (const double & other)

Matrix and number multiplication operator with assignment.

Parameters

other - number

◆ operator+() [1 / 2]

Matrix Matrix::operator+ (const double & other)

Addition operator of a matrix and a number.

Parameters

other - number

◆ operator+() [2 / 2]

Matrix Matrix::operator+ (const **Matrix** & other)

Operator for adding two matrices.

Parameters

other - matrix

◆ operator+=() [1/2]

Matrix & Matrix::operator+=(const double & other)

Addition operator with matrix and number assignment.

Parameters

other - number

◆ operator+=() [2/2]

Matrix & Matrix::operator+=(const **Matrix** & other)

Addition operator with assignment of two matrices.

Parameters

other - matrix

◆ operator-() [1/2]

Matrix Matrix::operator-(const double & other)

Difference operator of a matrix and a number.

Parameters

other - number

◆ operator-() [2/2]

Matrix Matrix::operator-(const **Matrix** & other)

Difference operator of two matrices.

Parameters

other - matrix

◆ operator-=() [1/2]

Matrix & Matrix::operator-=(const double & other)

The operator of the difference between the assignment matrix and the number.

Parameters

other - number

◆ operator-=() [2/2]

Matrix & Matrix::operator-=(const **Matrix** & other)

The difference operator of two matrices with assignment.

Parameters

other - matrix

◆ operator/()

Matrix Matrix::operator/ (const double & other)

Operator for dividing a matrix by a number.

Parameters

other - number

◆ operator/=()

Matrix & Matrix::operator/= (const double & other)

Operator for dividing a matrix by a number with assignment.

Parameters

other - number

◆ operator=()

Matrix & Matrix::operator= (const **Matrix** & other)

Matrix assignment operator.

Parameters

other - matrix

◆ operator==()

bool Matrix::operator== (const **Matrix** & other) const

Matrix comparison operator.

Parameters

other - matrix

◆ operator^()

Matrix Matrix::operator^ (const int & other)

The operator matrix of degree.

Parameters

other - number

◆ operator^=()

Matrix & Matrix::operator^= (const int & other)

Matrix exponentiation operator with assignment.

Parameters

other - number

◆ squareMatrixCheck()

```
bool Matrix::squareMatrixCheck ( ) const
```

Checks the matrix for squaring.

Returns

if the matrix is square - returns true
else - returns false

Friends And Related Function Documentation

◆ operator<<

```
ostream& operator<< ( ostream &    os,  
                    const Matrix & matrix  
                    )
```

friend

Output operator.

Returns

returns stream

◆ operator>>

```
istream& operator>> ( istream &    stream,  
                     Matrix &    matrix  
                     )
```

friend

Input operator.

Returns

returns stream

Вывод: В процессе выполнения данной лабораторной работы я изучил основы ООП на примере языка C++ , научился писать документацию в doxygen и тесты с использованием библиотеки catch.hpp, создал интерфейс для работы с программой. Результатом выполнения данной работы является программа для работы с матрицами (выполнения различных операций между: матрицей и числом, матрицей и матрицей, и вычисления различных параметров матрицы)