**Отчет по лабораторной работе №1**

**В – 14**

Разработать класс квадратная целочисленная матрица. Обязательны методы класса: ввода/вывода матрицы; вычисления следа матрицы; определителя; получение транспонированной матрицы; перестановка столбцов матрицы и строк; определение Евклидовой нормы матрицы.

Результат тестирования выводить на экран и в текстовый файл.

**ADT Matrix**

**Данные**

*int size размер массива*

**Операции**

*Конструктор*

Начальные значения: int size

Данные: Инициализация объекта

*set\_size(Задать размер матрицы)*

Вход: int size

Предусловия: Размер > 0

Процесс: Ввод размера матрицы

Выход: нет

Постусловия: изменился размер матрицы

*get\_size(Получить размер матрицы)*

Вход: нет

Предусловия: нет

Процесс: нет

Выход: size

Постусловия: нет

*create\_matrix(Создать матрицу)*

Вход: int size

Предусловия: нет

Процесс: Выделить память под массив

Выход: matr

Постусловия: Выделилась память под массив

*set\_matr(Задать элементы матрицы)*

Вход: int \*\*matr, int size

Предусловия: нет

Процесс: a[строка][столбец]= значение

Выход: нет

Постусловия: изменились элементы матрицы

*print\_matr(вывести матрицу на консоль)*

Вход: int \*\*matr, int size

Предусловия: нет

Процесс: Вывод матрицы в консоль

Выход: нет

Постусловия: нет

*sled\_matr(Получить след матрицы)*

Вход: int \*\*matr, int size

Предусловия: нет

Процесс: Расчет следа матрицы

Выход: sum

Постусловия: изменился след матрицы

*transpose\_matr(Транспонировать матрицу)*

Вход: int \*\*matr, int size

Предусловия: нет

Процесс: a[столбец][строка]= значение

Выход: нет

Постусловия: Создалась транспонирования матрица

ADT – абстрактный тип данных, создает определяемый пользователем тип данных.

Контейнер - тип, позволяющий инкапсулировать в себе объекты других типов. Контейнеры, в отличие от коллекций, реализуют конкретную структуру данных.

**Файл Matr.h**

#pragma once

class Matrix

{

private:

int size;

int\* matr;

public:

Matrix();

Matrix(int size);

void set\_size(int size1);

int get\_size();

void set\_matr(int\*\* matr1, int size1);

int\*\* create\_matrix(int size1);

void print\_matrix(int\*\* matr1, int size1);

int sled\_matr(int\*\* matr1, int size1);

void transpose\_matr1(int\*\* matr1, int size1);

};

**Файл Matr.cpp**

///Подключаем заголовочный файл

#include "Matr.h"

/// Библиотека для ввода и вывода

#include <iostream>

/// Библиотека для форматирования строк

#include <iomanip>

#include <string>

using namespace std;

Matrix::Matrix()

{

size = 0;

matr = nullptr;

}

Matrix::Matrix(int size1)

{

size = size1;

matr = new int[size];

}

void Matrix::set\_size(int size1)

{

if (size1 == 0) throw std::invalid\_argument("Error size = 0");

size = size1;

}

int Matrix::get\_size() { return size; }

int\*\* Matrix::create\_matrix(int size1)

{

int\*\* matr = new int\* [size1];

for (int i = 0; i < size1; i++)

{

matr[i] = new int[size1];

}

return matr;

}

void Matrix::set\_matr(int\*\* matr1, int size1)

{

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size1; j++)

{

cout << "[" << i << "]" << "[" << j << "] = ";

cin >> matr1[i][j];

}

}

}

void Matrix::print\_matrix(int\*\* matr1, int size1)

{

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size1; j++)

{

cout << matr1[i][j] << " ";

}

cout << endl;

}

}

int Matrix::sled\_matr(int \*\*matr1, int size1)

{

int sum = 0;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size1; j++)

{

if (i == j)

sum += matr1[i][j];

}

}

return sum;

}

void Matrix::transpose\_matr1(int\*\* matr1,int size1)

{

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

{

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

{

//matr1[j][i];

cout << matr1[j][i] << " ";

}

cout << endl;

}

}

**Файл MainMatr.cpp**

#include <iostream>

#include "Matr.h"

#include <fstream>

using namespace std;

void printInFile(string file\_name, int\*\* matr1, int size1, int sled)

{

ofstream f(file\_name);

if (file\_name == "") throw std::invalid\_argument("File's name error"); /// Если файл без имени

if (!f.is\_open()) throw std::invalid\_argument("File isn't open!"); /// Если файл не открыт

f << "Изначальная матрица: " << endl;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size1; j++)

{

f << matr1[i][j] << " ";

}

f << endl;

}

f << "След матрицы = " << sled;

f << endl;

f << "Транспонированная матрица: " << endl;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size1; j++)

{

f << matr1[j][i] << " ";

}

f << endl;

}

f.close();

}

int main()

{

setlocale(LC\_ALL, "Russian");

try {

Matrix m1;

Matrix m2;

int Msize = 0;

cout << "Введите размер матрицы = ";

cin >> Msize;

m1.set\_size(Msize);

int\*\* Matrix1 = m1.create\_matrix(Msize);

cout << "Введите элементы матрицы: " << endl;

m1.set\_matr(Matrix1, Msize);

m1.print\_matrix(Matrix1, Msize);

int sled = m1.sled\_matr(Matrix1, Msize);

cout << "След матрицы = " << sled << endl;

cout << "Транспонированная матрицы: " << endl;

m1.transpose\_matr1(Matrix1,Msize);

cout << endl;

printInFile("test1.txt", Matrix1, Msize, m1.sled\_matr(Matrix1,Msize));

}

catch (invalid\_argument e)

{

cout << e.what();

}

}



