ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ПРАКТИЧЕСКОЙ РАБОТЕ № 2

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Райхман И. А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Содержание**

Задание 23

Задание 1

## Формулировка задачи

Реализовать задание 4.3 из прошлого семестра в ООП стиле.

Создать решение (Solution), которое минимально содержит три проекта (Projects): исполняемый (Console Application), библиотеку классов (Library), и модульные тесты (Tests). Разработать библиотеку классов по заданному варианту. Важно! Библиотека классов не должна зависеть от потоков ввода-вывода. Каждый класс необходимо размещать в отдельных *двух* файлах, снабжённых «говорящим именем» и специальными расширениями: \*.h для заголовочных файлов (Header), содержащих API класса, и \*.cpp для компилируемых (Source), содержащих реализацию класса. В запускаемом проекте требуется создать файл main.cpp, содержащий точку входа в демонстрационную программу – главную функцию (main). В рамках данной функции показать работу с массивом.

Создать класс Matrix (двумерный массив) (можно использовать std::vector<T>). Реализовать все конструкторы, создаваемые компилятором по умолчанию, реализовать деструктор. Предусмотреть методы вывода в строку содержимого массива. Переопределить операторы присваивания, сдвига влево и сдвига вправо. Переопределить оператор разыменования элемента коллекции по индексу.

Предусмотреть заполнение массива по определенному алгоритму (случайным образом, вводом с клавиатуры, заполнение нулями и константным значением) реализовать через класс Generator, конкретный алгоритм реализовать в классе наследнике.

## UML - диаграмма алгоритма

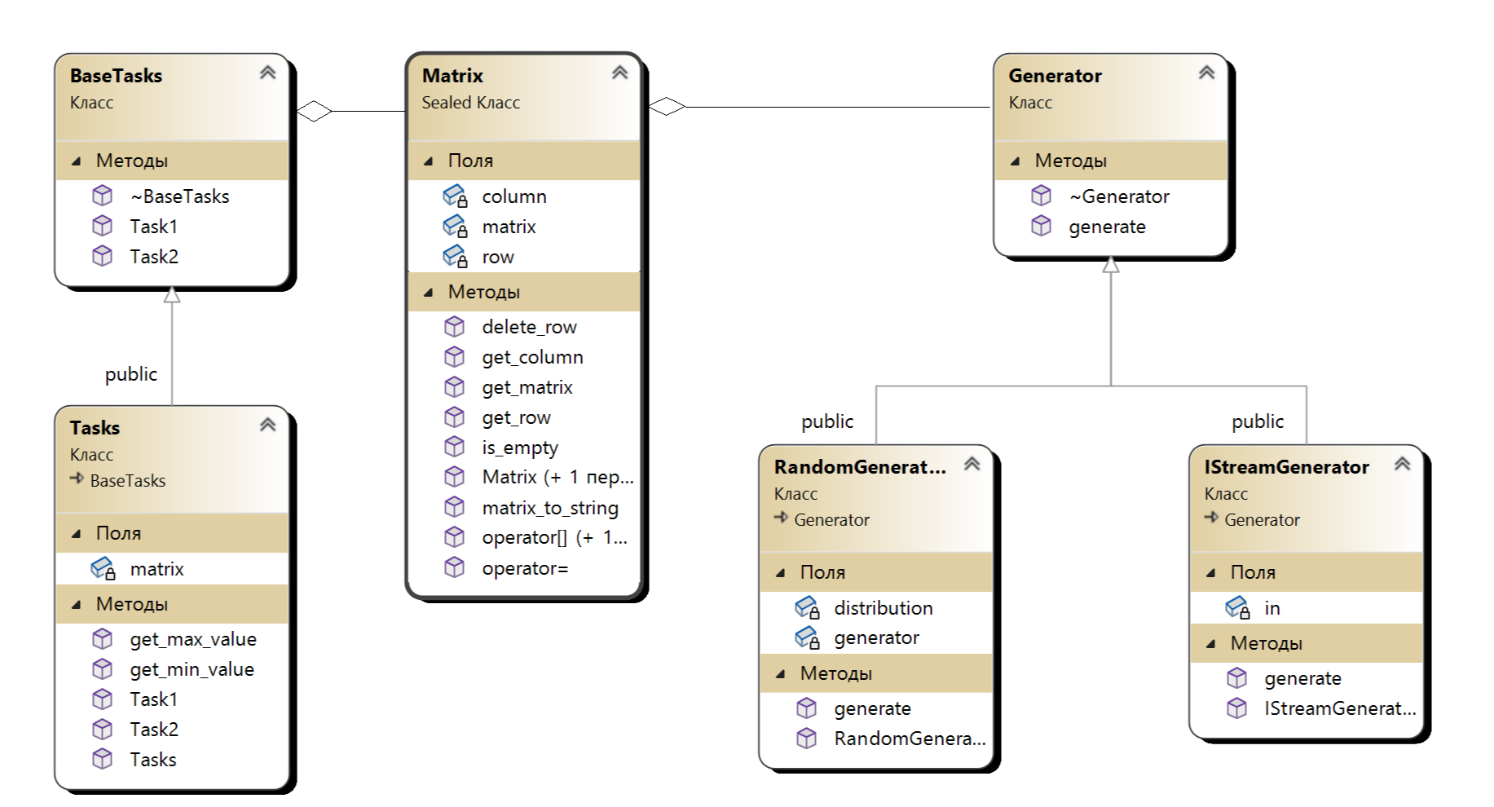


Рисунок 1 – UML диаграмма класса.

## Решение задачи на языке программирования C++

main.cpp:

#include "../Matrix/Matrix.h"

#include "../Matrix/RandomGenerator.h"

#include "../Matrix/IStreamGenerator.h"

#include "../Matrix/Tasks.h"

#include <iostream>

/\*\*

\* @brief Точка входа в программу

\* @return Возвращает 0 в случае успеха

\*/

int main()

{

setlocale(LC\_ALL, "Rus");

RandomGenerator rnd(-10, 10);

//std::istream& in = std::cin;

//IStreamGenerator rnd(in);

Matrix matrix(3, 3, &rnd);

std::cout << matrix << std::endl;

Tasks number\_one(matrix);

std::cout << number\_one.Task1().matrix\_to\_string();

std::cout << "\n";

Tasks number\_two(matrix);

std::cout << number\_two.Task2().matrix\_to\_string();

return 0;

}

Matrix.h:

#pragma once

#include "BaseGenerator.h"

#include <iostream>

#include <vector>

#include <sstream>

#include <exception>

/\*\*

\* @brief класс Матрицы

\*/

class Matrix final

{

public:

/\*\*

\* @brief Конструктор

\* @param row Количество строк

\* @param col Количество столбцов

\*/

Matrix(const size\_t row, const size\_t column, Generator\* generator);

/\*\*

\* @brief Конструктор копирования

\* @param other Матрица для копирования

\*/

Matrix(const Matrix& other) = default;

/\*\*

\* @brief Получение кол-ва строк

\* @return Количество строк

\*/

size\_t get\_row() const;

/\*\*

\* @brief Получение кол-ва столбцов

\* @return Количество столбцов

\*/

size\_t get\_column() const;

/\*\*

\* @brief Проверка на пустоту матрицы

\* @return Логическое значение

\*/

bool is\_empty() const;

/\*\*

\* @brief Переопределение оператора []

\* @param index Индекс матрицы

\* @return Элемент матрицы по индексу

\*/

std::vector<int>& operator[](size\_t index);

/\*\*

\* @brief Переопределение константного оператора []

\* @param index Индекс матрицы

\* @return Элемент матрицы по индексу

\*/

const std::vector<int>& operator[](size\_t index) const;

/\*\*

\* @brief Переопределение оператора присваивания

\* @param other Матрица для копирования

\* @return Элемент матрицы по индексу

\*/

Matrix& operator=(const Matrix& other) = default;

/\*\*

\* @brief Переопределение оператора вывода

\* @param os Поток вывода

\* @param matrix Матрица

\* @return Элемент матрицы по индексу

\*/

friend std::ostream& operator<<(std::ostream& os, Matrix& matrix);

/\*\*

\* @brief Вывод матрицы в строку

\* @return Матрица в строку

\*/

std::string matrix\_to\_string() const;

/\*\*

\* @brief Удаление строки

\*/

void delete\_row(size\_t pos, const std::vector<int>& row);

/\*\*

\* @brief Получение матрицы

\* @return Матрица

\*/

std::vector<std::vector<int>> get\_matrix();

private:

size\_t column;

size\_t row;

std::vector<std::vector<int>> matrix;

};

Matrix.cpp:

#include "Matrix.h"

Matrix::Matrix(const size\_t row, const size\_t column, Generator\* generator)

: row(row), column(column)

{

if (column < 0 || row < 0)

{

throw std::invalid\_argument("Неправильный размер матрицы :(");

}

if (generator == nullptr)

{

throw std::invalid\_argument("Ошибка генератора матрицы :(");

}

std::vector<std::vector<int>> matrix(row, std::vector<int>(column));

for (size\_t i = 0; i < row; i++)

{

for (size\_t j = 0; j < column; j++)

{

matrix[i][j] = generator->generate();

}

}

this->matrix = matrix;

}

size\_t Matrix::get\_row() const

{

return matrix.size();

}

size\_t Matrix::get\_column() const

{

if (is\_empty())

{

return 0;

}

return matrix[0].size();

}

bool Matrix::is\_empty() const

{

return (matrix.size() == 0) || (column == 0);

}

std::vector<int>& Matrix::operator[](size\_t index)

{

return matrix[index];

}

const std::vector<int>& Matrix::operator[](size\_t index) const

{

if (index < 0)

{

throw std::out\_of\_range("Ошибка! Неправильный индекс");

}

return matrix[index];

}

std::string Matrix::matrix\_to\_string() const

{

std::stringstream buffer;

for (size\_t i = 0; i < row; i++)

{

for (size\_t j = 0; j < column; j++)

{

buffer << matrix[i][j] << ' ';

}

buffer << std::endl;

}

return buffer.str();

}

void Matrix::delete\_row(size\_t pos, const std::vector<int>& row)

{

if (pos >= matrix.size())

{

throw std::out\_of\_range("Error! Invalid position.");

}

auto it = matrix.begin() + pos;

if (std::equal(it->begin(), it->end(), row.begin()))

{

matrix.erase(it);

}

else

{

throw std::invalid\_argument("Error! Row does not match at the specified position.");

}

}

std::vector<std::vector<int>> Matrix::get\_matrix()

{

return this->matrix;

}

std::ostream& operator<<(std::ostream& os, Matrix& matrix)

{

for (size\_t i = 0; i < matrix.row;i++)

{

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

{

os << matrix[i][j] << " ";

}

os << std::endl;

}

return os;

}

BaseTasks.h:

#pragma once

#include "Matrix.h"

#include "BaseGenerator.h"

/\*\*

\* @brief Абстрактный класс заданий

\*/

class BaseTasks

{

public:

/\*\*

\* @brief Виртуальный деструткор

\*/

virtual ~BaseTasks() = 0 {};

/\*\*

\* @brief Виртуальный метод для первого задания

\*/

virtual Matrix Task1() = 0;

/\*\*

\* @brief Виртуальный метод для второго задания

\*/

virtual Matrix Task2() = 0;

};

Tasks.h:

#pragma once

#include "Matrix.h"

#include "BaseTasks.h"

/\*\*

\* @brief класс Tasks.

\*/

class Tasks : public BaseTasks

{

public:

/\*\*

\* @brief конструктор

\*/

Tasks(Matrix matrix);

/\*\*

\* @brief метод для Задания 1

\* @return матрицу

\*/

Matrix Task1() override;

/\*\*

\* @brief метод для Задания 2

\* @return матрицу

\*/

Matrix Task2() override;

/\*\*

\* @brief метод для получения минимального значения

\* @return минимальное значение

\*/

int get\_min\_value();

/\*\*

\* @brief метод для получения максимального значения

\* @return максимальное значение

\*/

int get\_max\_value();

private:

Matrix matrix;

};

Tasks.cpp:

#include "Tasks.h"

Tasks::Tasks(Matrix matrix)

: matrix(matrix)

{

}

Matrix Tasks::Task1()

{

for (size\_t i = 0; i < matrix.get\_column(); i++)

{

int min\_value = get\_min\_value();

for (size\_t j = 0; j < matrix.get\_column(); j++)

{

if (matrix[j][i] == min\_value)

{

matrix[j][i] = abs(matrix[j][i]);

}

}

}

return matrix;

}

Matrix Tasks::Task2()

{

int max\_value = this->matrix[0][0];

size\_t row\_index = 0;

for (size\_t i = 0; i < this->matrix.get\_matrix().size(); i++)

{

for (size\_t j = 0; j < this->matrix[i].size(); j++)

{

if (this->matrix[i][j] > max\_value)

{

max\_value = this->matrix[i][j];

row\_index = i;

}

}

}

Matrix new\_matrix = matrix;

this->matrix = new\_matrix;

return new\_matrix;

}

int Tasks::get\_min\_value()

{

int min\_value = matrix[0][0];

for (size\_t i = 0; i < matrix.get\_column(); i++)

{

for (size\_t j = 1; j < matrix.get\_row(); j++)

{

if (matrix[j][i] < min\_value)

{

min\_value = matrix[j][i];

}

}

}

return min\_value;

}

int Tasks::get\_max\_value()

{

int max\_value = 0;

for (size\_t i = 0; i < matrix.get\_column(); i++)

{

max\_value = matrix[0][i];

for (size\_t j = 1; j < matrix.get\_row(); j++)

{

if (matrix[j][i] > max\_value)

{

max\_value = matrix[j][i];

}

}

}

return max\_value;

}

BaseGenerator.h:

#pragma once

/\*\*

\* @brief базовый класс Generator

\*/

class Generator

{

public:

/\*\*

\* @brief Виртуальный деструктор

\*/

virtual ~Generator() = 0 {};

/\*\*

\* @brief Виртуальный метод генератор

\*/

virtual int generate() = 0;

};

IStreamGenerator.h:

#pragma once

#include "BaseGenerator.h"

#include <iostream>

/\*\*

\* @brief Класс IStreamGenerator

\*/

class IStreamGenerator : public Generator

{

private:

std::istream& in;

public:

/\*\*

\* @brief Конструктор класса

\* @param in поток ввода

\*/

IStreamGenerator(std::istream& in = std::cin);

/\*\*

\* @brief метод заполения generate()

\*/

int generate() override;

};

IStreamGenerator.cpp:

#include "IStreamGenerator.h"

IStreamGenerator::IStreamGenerator(std::istream& in) : in{ in }

{

}

int IStreamGenerator::generate()

{

int value = 0;

this->in >> value;

return value;

}

RandomGenerator.h:

#pragma once

#include "BaseGenerator.h"

#include <random>

/\*\*

\* @brief класс RandomGenerator

\*/

class RandomGenerator : public Generator

{

private:

std::uniform\_int\_distribution<int> distribution;

std::mt19937 generator;

public:

/\*\*

\* @brief Метод рандомного заполнения

\* @param min минимальный элемент

\* @param max максимальный элемент

\*/

RandomGenerator(const int min, const int max);

/\*\*

\* @brief Метод заполения generate()

\*/

int generate() override;

};

## Решение тестов

Код тестов:

#include "CppUnitTest.h"

#include "../Matrix/Matrix.h"

#include "../Matrix/BaseGenerator.h"

#include "../Matrix/RandomGenerator.h"

#include "../Matrix/IStreamGenerator.h"

#include "../Matrix/Tasks.h"

#include <iostream>

#include <sstream>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace MatrixTests

{

TEST\_CLASS(MatrixTests)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Assert::IsNotNull(&matrix);

}

TEST\_METHOD(CopyConstructorTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Matrix second\_matrix(matrix);

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(matrix.matrix\_to\_string(), second\_matrix.matrix\_to\_string());

Assert::AreNotSame(matrix.matrix\_to\_string(), second\_matrix.matrix\_to\_string());

}

TEST\_METHOD(GetRowTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

size\_t expected\_output = 3;

Matrix matrix(3, 3, &rnd);

Assert::AreEqual(expected\_output, matrix.get\_row());

}

TEST\_METHOD(GetColumnTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

size\_t expected\_output = 3;

Matrix matrix(3, 3, &rnd);

Assert::AreEqual(expected\_output, matrix.get\_column());

}

TEST\_METHOD(IndexOperatorTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 9");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

int expected\_output = 1;

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(expected\_output, matrix[0][0]);

}

TEST\_METHOD(ConstIndexOperatorTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 9");

IStreamGenerator input(in);

const Matrix matrix(3, 3, &input);

int expected\_output = 1;

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(expected\_output, matrix[0][0]);

}

TEST\_METHOD(CopyOperatorTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Matrix second\_matrix(3, 3, &rnd);

second\_matrix = matrix;

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(matrix.matrix\_to\_string(), second\_matrix.matrix\_to\_string());

Assert::AreNotSame(matrix.matrix\_to\_string(), second\_matrix.matrix\_to\_string());

}

TEST\_METHOD(OutputOperatorTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 9");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

std::stringstream buffer;

std::string expected\_output = "1 2 3 \n4 5 6 \n7 8 9 \n";

buffer << matrix;

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(expected\_output, buffer.str());

}

TEST\_METHOD(IsEmptyTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Assert::IsFalse(matrix.is\_empty());

}

TEST\_METHOD(MatrixToStringTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 9");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

std::string expected\_output = "1 2 3 \n4 5 6 \n7 8 9 \n";

Assert::IsFalse(matrix.is\_empty());

Assert::AreEqual(expected\_output, matrix.matrix\_to\_string());

}

};

TEST\_CLASS(TasksTests)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Tasks task(matrix);

Assert::IsNotNull(&task);

}

TEST\_METHOD(MinValueTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 9");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

std::stringstream buffer;

std::string expected\_output = "1";

Tasks task(matrix);

buffer << task.get\_min\_value();

Assert::AreEqual(expected\_output, buffer.str());

}

TEST\_METHOD(TaskOneTest\_ValidData\_Success)

{

std::istringstream in("1 2 3 4 5 6 7 8 -9");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

Tasks task(matrix);

std::string expected\_output = "1 2 3 \n4 5 6 \n7 8 9 \n";

Assert::AreEqual(task.Task1().matrix\_to\_string(), expected\_output);

}

TEST\_METHOD(TaskTwoTest\_ValidData\_Success)

{

std::istringstream in("-3 8 6 -3 -10 8 0 -6 -1");

IStreamGenerator input(in);

Matrix matrix(3, 3, &input);

Tasks task(matrix);

std::string expected = "-3 8 6 \n-3 -10 8 \n0 -6 -1 \n";

Assert::AreEqual(task.Task2().matrix\_to\_string(), expected);

}

};

}

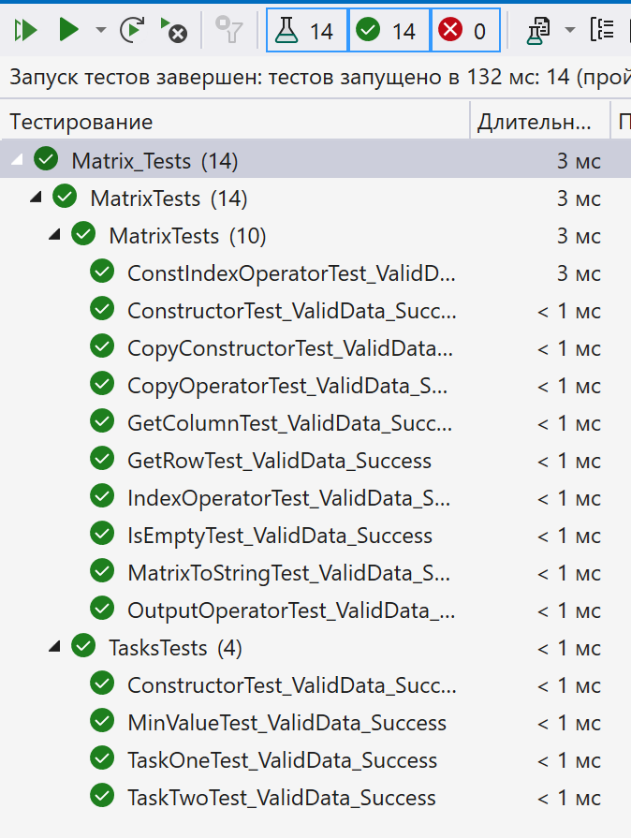


Рисунок 2 – Успешные тесты

## Зачет задания в GitHub



Рисунок 3 - Зачет задания в GitHub