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

Захаров Дмитрий Константинович

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Задание:** написать класс матрица

**Код задания:**

#include "Matrix.h"

#include <random>

#include <sstream>

miit::Matrix::Matrix(size\_t user\_rows, size\_t user\_columns, Generator\* generator)

:data({})

{

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

{

std::vector<int> temp{};

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

{

temp.push\_back(generator->generate());

}

this->data.push\_back(temp);

}

}

miit::Matrix::Matrix(size\_t user\_rows, size\_t user\_columns, std::initializer\_list<int> data)

:data(std::vector<std::vector<int>> {user\_rows})

{

size\_t i = 0;

size\_t j = 0;

for (auto& item : data)

{

this->data[j].push\_back(item);

i++;

if (i % user\_columns == 0)

{

j++;

}

}

}

miit::Matrix::Matrix()

:data(std::vector<std::vector<int>> {})

{

}

void miit::Matrix::add\_row(std::vector<int> user\_data, size\_t index)

{

if (index < 0 || index > this->rows\_counts())

{

throw std::out\_of\_range("incorrect index");

}

this->data.insert(data.begin() + index, user\_data);

}

void miit::Matrix::add\_row(std::vector<int> user\_data)

{

this->data.push\_back(user\_data);

}

void miit::Matrix::add\_column(std::vector<int> user\_data, size\_t index)

{

transpose();

this->add\_row(user\_data, index);

transpose();

}

void miit::Matrix::add\_column(std::vector<int> user\_data)

{

transpose();

this->add\_row(user\_data);

transpose();

}

int miit::Matrix::get\_abs\_max\_element()

{

int max = 0;

for (size\_t i = 0; i < this->rows\_counts(); i++)

{

for (size\_t j = 0; j < this->columns\_counts(); j++)

{

if (std::abs(this->data[i][j]) >= std::abs(max))

{

max = this->data[i][j];

}

}

}

return max;

}

bool miit::Matrix::is\_element\_in\_row(int value, size\_t row\_id)

{

for (size\_t i = 0; i < this->data[row\_id].size(); i++)

{

if (this->data[row\_id][i] == value)

{

return true;

}

}

return false;

}

size\_t miit::Matrix::rows\_counts() const

{

return this->data.size();

}

size\_t miit::Matrix::columns\_counts() const

{

if (this->data.size() == 0)

{

return 0;

}

return this->data[0].size();

}

void miit::Matrix::transpose()

{

std::vector<std::vector<int>> temp\_data;

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

{

std::vector<int> temp {};

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

{

temp.push\_back(this->data[i][j]);

}

temp\_data.push\_back(temp);

}

this->data = temp\_data;

}

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

{

return data[index];

}

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

{

return data[index];

}

std::string miit::Matrix::to\_string() const noexcept

{

std::stringstream temp;

for (size\_t i = 0; i < this->rows\_counts(); i++)

{

for (size\_t j = 0; j < this->columns\_counts(); j++)

{

temp << this->data[i][j] << "\t";

}

temp << "\n";

}

return temp.str();

}

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

{

return os << matrix.to\_string();

}

bool miit::operator==(Matrix& lha, Matrix& rha) noexcept

{

return lha.to\_string() == rha.to\_string();

}

#pragma once

#include "../Generator/Generator.h"

#include <iostream>

#include <vector>

namespace miit

{

/\*\*

\* @brief Класс описывающий матрицу

\*/

class Matrix

{

public:

/\*\*

\* @brief Инициализирует объект тип матрица по параметрам пользователя

\* @param user\_rows Количество строк в матрице

\* @param user\_columns Количество столбцов в матрице

\* @param generator Генератор, которым необходимо заполнить матрицу

\*/

Matrix(size\_t user\_rows, size\_t user\_columns, Generator\* generator);

/\*\*

\* @brief Инициализирует объект тип матрица по параметрам пользователя

\* @param user\_rows Количество строк в матрице

\* @param user\_columns Количество столбцов в матрице

\* @param data Информация о значениях матрицы

\*/

Matrix(size\_t user\_rows, size\_t user\_columns, std::initializer\_list<int> data);

/\*\*

\* @brief Инициализирует пустой объект тип матрица

\*/

Matrix();

/\*\*

\* @brief Добавляет строку в матрицу по индексу, если индекс не указан, то добавит элемент в конец

\* @param user\_data Строка для добавления

\* @param index Индекс на который строка будет добавлена

\*/

void add\_row(std::vector<int> user\_data, size\_t index);

/\*\*

\* @brief Добавляет строку в матрицу по индексу, если индекс не указан, то добавит элемент в конец

\* @param user\_data Строка для добавления

\* @param index Индекс на который строка будет добавлена

\*/

void add\_row(std::vector<int> user\_data);

/\*\*

\* @brief Добавляет столбец в матрицу по индексу, если индекс не указан, то добавит элемент в конец

\* @param user\_data Столбец для добавления

\* @param index Индекс на который столбец будет добавлен

\*/

void add\_column(std::vector<int> user\_data, size\_t index);

/\*\*

\* @brief Добавляет столбец в матрицу по индексу, если индекс не указан, то добавит элемент в конец

\* @param user\_data Столбец для добавления

\* @param index Индекс на который столбец будет добавлен

\*/

void add\_column(std::vector<int> user\_data);

/\*\*

\* @brief Получение максимального по модолю элемента в матрице

\* @return Максимальный по модолю елемент в матрице

\*/

int get\_abs\_max\_element();

/\*\*

\* @brief Проверка на наличие элемента в строке матрицы

\* @return true - если есть в сроке, false - если нет

\*/

bool is\_element\_in\_row(int value, size\_t row\_id);

/\*\*

\* @brief Получение количества строк в матрице

\* @return Количество строк в матрице

\*/

size\_t rows\_counts() const;

/\*\*

\* @brief Получение количества столбцов в матрице

\* @return Количество столбцов в матрице

\*/

size\_t columns\_counts() const;

/\*\*

\* @brief Транспонирование матрицы

\*/

void transpose();

/\*\*

\* @brief Перегрузка оператора вывода

\* @param os Поток вывода

\* @param matrix Матрица, данные которой будут выведены в поток

\* @return Поток вывода с информацией матрицы

\*/

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

/\*\*

\* @brief Перегрузка оператора сравнения

\* @param lha Первый аргумент для сравнения

\* @param rha Второй аргумент для сравнения

\* @return true - если равны, false - если нет

\*/

friend bool operator==(Matrix& lha, Matrix& rha) noexcept;

/\*\*

\* @brief Перегрузка оператора квадратные скобки

\* @param index Индекс элемента в массиве

\* @return Элемент из матрцы стоящий под нужным индексом

\*/

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

/\*\*

\* @brief Перегрузка оператора квадратные скобки для объекта с const

\* @param index Индекс элемента в массиве

\* @return Элемент из матрцы стоящий под нужным индексом

\*/

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

/\*\*

\* @brief Преобразование объекта матрицы в строку

\* @return Строка с информацией о матрице

\*/

std::string to\_string() const noexcept;

private:

std::vector<std::vector<int>> data;

};

};

#include "Tasks.h"

miit::Tasks::Tasks(Matrix matrix)

:matrix(matrix)

{

}

miit::Matrix miit::Tasks::resolve\_task\_1()

{

miit::Matrix result{this->matrix};

int max = this->matrix.get\_abs\_max\_element();

for (size\_t i = 0; i < result.rows\_counts(); i++)

{

for (size\_t j = 0; j < result.columns\_counts(); j++)

{

if (result[i][j] == max)

{

result[i][j] = 0;

}

}

}

return result;

}

miit::Matrix miit::Tasks::resolve\_task\_2()

{

miit::Matrix result {};

std::vector<int> zero\_row;

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

{

zero\_row.push\_back(0);

}

int max = this->matrix.get\_abs\_max\_element();

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

{

result.add\_row(matrix[i]);

if (this->matrix.is\_element\_in\_row(max, i))

{

result.add\_row(zero\_row);

}

}

return result;

}

#pragma once

#include "BaseTask.h"

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

namespace miit

{

/\*\*

\* @brief Класс задание, что включает в себя 2 подзадания

\*/

struct Tasks : public BaseTask

{

private:

Matrix matrix;

public:

/\*\*

\* @brief Инициализация объекта типа Tasks

\* @param matrix Матрица с которой надо выполнить задания

\*/

Tasks(Matrix matrix);

/\*\*

\* @brief Решение первого задания

\* @return Матрица с данными выполенного задания

\*/

Matrix resolve\_task\_1() override;

/\*\*

\* @brief Решение второго задания

\* @return Матрица с данными выполенного задания

\*/

Matrix resolve\_task\_2() override;

};

}

#pragma once

/\*\*

\* @brief Базовый класс генератора

\*/

class Generator

{

public:

/\*\*

\* @brief Чисто виртуальный деструктор

\*/

virtual ~Generator() = 0 {};

/\*\*

\* @brief Чисто виртуальный метод для генерации числа в классах наследниках

\*/

virtual int generate() = 0;

};

#pragma once

#include <random>

#include "Generator.h"

/\*\*

\* @brief Генератор случайного числа в заданном диапазоне

\*/

class RandomGenerator : public Generator

{

private:

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

std::mt19937 generator;

public:

/\*\*

\* @brief Инициализация объекта типа RandomGenerator

\* @param min Минимальное значение для рандома

\* @param max Максимальное значение для рандома

\*/

RandomGenerator(int min, int max);

/\*\*

\* @brief Генерирует рандомное число из заданного ранее диапазона

\* @return Случайные элемент из заданного диапазона

\*/

int generate() override;

};

#include "RandomGenerator.h"

RandomGenerator::RandomGenerator(int min, int max)

{

this->generator = std::mt19937(std::random\_device{}());

this->distribution = std::uniform\_int\_distribution<int>(min, max);

}

int RandomGenerator::generate()

{

return this->distribution(this->generator);

}

#include "RandomGenerator.h"

RandomGenerator::RandomGenerator(int min, int max)

{

this->generator = std::mt19937(std::random\_device{}());

this->distribution = std::uniform\_int\_distribution<int>(min, max);

}

int RandomGenerator::generate()

{

return this->distribution(this->generator);

}

#pragma once

#include <random>

#include "Generator.h"

/\*\*

\* @brief Генератор случайного числа в заданном диапазоне

\*/

class RandomGenerator : public Generator

{

private:

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

std::mt19937 generator;

public:

/\*\*

\* @brief Инициализация объекта типа RandomGenerator

\* @param min Минимальное значение для рандома

\* @param max Максимальное значение для рандома

\*/

RandomGenerator(int min, int max);

/\*\*

\* @brief Генерирует рандомное число из заданного ранее диапазона

\* @return Случайные элемент из заданного диапазона

\*/

int generate() override;

};

**Тесты:**

#include "pch.h"

#include "CppUnitTest.h"

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

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

using namespace miit;

namespace MatrixTests

{

TEST\_CLASS(MatrixTests)

{

public:

TEST\_METHOD(MatrixToString\_ValidData\_Success)

{

Matrix matrix{ 4,2, {1,2,3,4,5,6,7,8} };

Assert::IsTrue(matrix.to\_string() == "1\t2\t\n3\t4\t\n5\t6\t\n7\t8\t\n");

}

TEST\_METHOD(MatrixAddRow\_ValidData\_Success)

{

Matrix matrix{ 3,3, {1,1,1,1,1,1,1,1,1} };

Matrix success\_matrix = { 4,3, {1,1,1,1,1,1,1,1,1,5,5,5} };

std::vector<int> temp {5, 5, 5};

matrix.add\_row(temp);

Assert::IsTrue(success\_matrix == matrix);

}

TEST\_METHOD(MatrixAddRowByIndex\_ValidData\_Success)

{

Matrix matrix{ 3,3, {1,1,1, 1,1,1, 1,1,1} };

Matrix success\_matrix = { 4,3, {1,1,1, 5,5,5, 1,1,1, 1,1,1} };

std::vector<int> temp{ 5, 5, 5 };

matrix.add\_row(temp, 1);

Assert::IsTrue(success\_matrix == matrix);

}

TEST\_METHOD(MatrixAddColumn\_ValidData\_Success)

{

Matrix matrix{ 3,3, {1,1,1, 1,1,1, 1,1,1} };

Matrix success\_matrix = { 3,4, {1,1,1,5, 1,1,1,5, 1,1,1,5} };

std::vector<int> temp{ 5, 5, 5 };

matrix.add\_column(temp);

Assert::IsTrue(success\_matrix == matrix);

}

TEST\_METHOD(MatrixAddColumnByIndex\_ValidData\_Success)

{

Matrix matrix{ 3,3, {1,1,1, 1,1,1, 1,1,1} };

Matrix success\_matrix = { 3,4, {1,5,1,1, 1,5,1,1, 1,5,1,1} };

std::vector<int> temp{ 5, 5, 5 };

matrix.add\_column(temp, 1);

Assert::IsTrue(success\_matrix == matrix);

}

TEST\_METHOD(MatrixGetMaxAbsElement\_ValidData\_Success)

{

Matrix matrix{ 3,3, {22, 1, -30, 7, 23, 5, -3, 4, -12} };

int success\_value = -30;

int temp = matrix.get\_abs\_max\_element();

Assert::IsTrue(temp == success\_value);

}

TEST\_METHOD(MatrixIsElementInRow\_ValidData\_Success)

{

Matrix matrix{ 3,3, {32, 1, -30, 7, 23, 5, -3, 4, -12} };

int temp\_1 = matrix.is\_element\_in\_row(23,1);

int temp\_2 = matrix.is\_element\_in\_row(23, 0);

Assert::IsTrue(temp\_1 == true && temp\_2 == false);

}

TEST\_METHOD(MatrixGetRowsCount\_ValidData\_Success)

{

Matrix matrix{ 3,3, {32, 1, -30, 7, 23, 5, -3, 4, -12} };

int temp\_1 = matrix.is\_element\_in\_row(23, 1);

int temp\_2 = matrix.is\_element\_in\_row(23, 0);

Assert::IsTrue(temp\_1 == true && temp\_2 == false);

}

TEST\_METHOD(MatrixRowsCounts\_ValidData\_Success)

{

Matrix matrix{ 3,3, {32, 1, -30, 7, 23, 5, -3, 4, -12} };

int temp = matrix.rows\_counts();

Assert::IsTrue(temp == 3);

}

TEST\_METHOD(MatrixColumnsCounts\_ValidData\_Success)

{

Matrix matrix{ 3,3, {32, 1, -30, 7, 23, 5, -3, 4, -12} };

int temp = matrix.columns\_counts();

Assert::IsTrue(temp == 3);

}

TEST\_METHOD(MatrixTranspose\_ValidData\_Success)

{

Matrix matrix{ 2,4, {1,2,3,4,5,6,7,8} };

matrix.transpose();

Assert::IsTrue(matrix.to\_string() == "1\t5\t\n2\t6\t\n3\t7\t\n4\t8\t\n");

}

TEST\_METHOD(MatrixEqualOperator\_ValidData\_Success)

{

Matrix matrix\_1{ 4,2, {1,2,3,4,5,6,7,8} };

Matrix matrix\_2{ 4,2, {1,2,3,4,5,6,7,8} };

Matrix matrix\_3{ 4,2, {0,-2,2,4,12,6,4,8} };

Assert::IsTrue(matrix\_1 == matrix\_2 && !(matrix\_1 == matrix\_3));

}

TEST\_METHOD(MatrixIndexOperator\_ValidData\_Success)

{

Matrix matrix{ 4,2, {1,2,3,4,5,6,7,8} };

int success = 3;

Assert::IsTrue(matrix[1][0] == success);

}

};

}

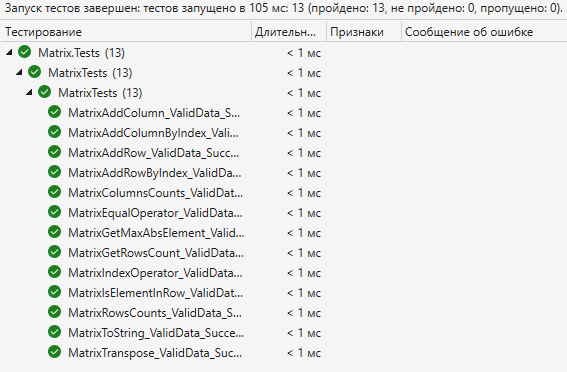
****

Рисунок 1 – прохождение тестов

**UML Диаграмма:**

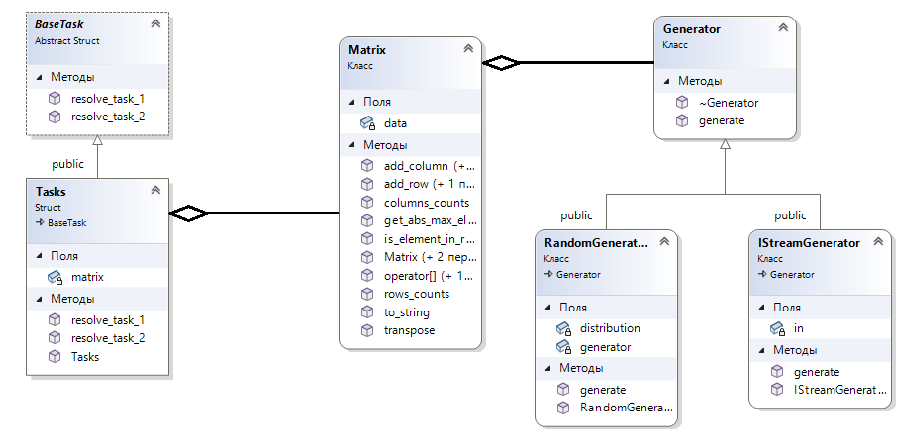


Рисунок 2 – UML диаграмма



Рисунок 3 – Approve