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

Москалев П.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

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

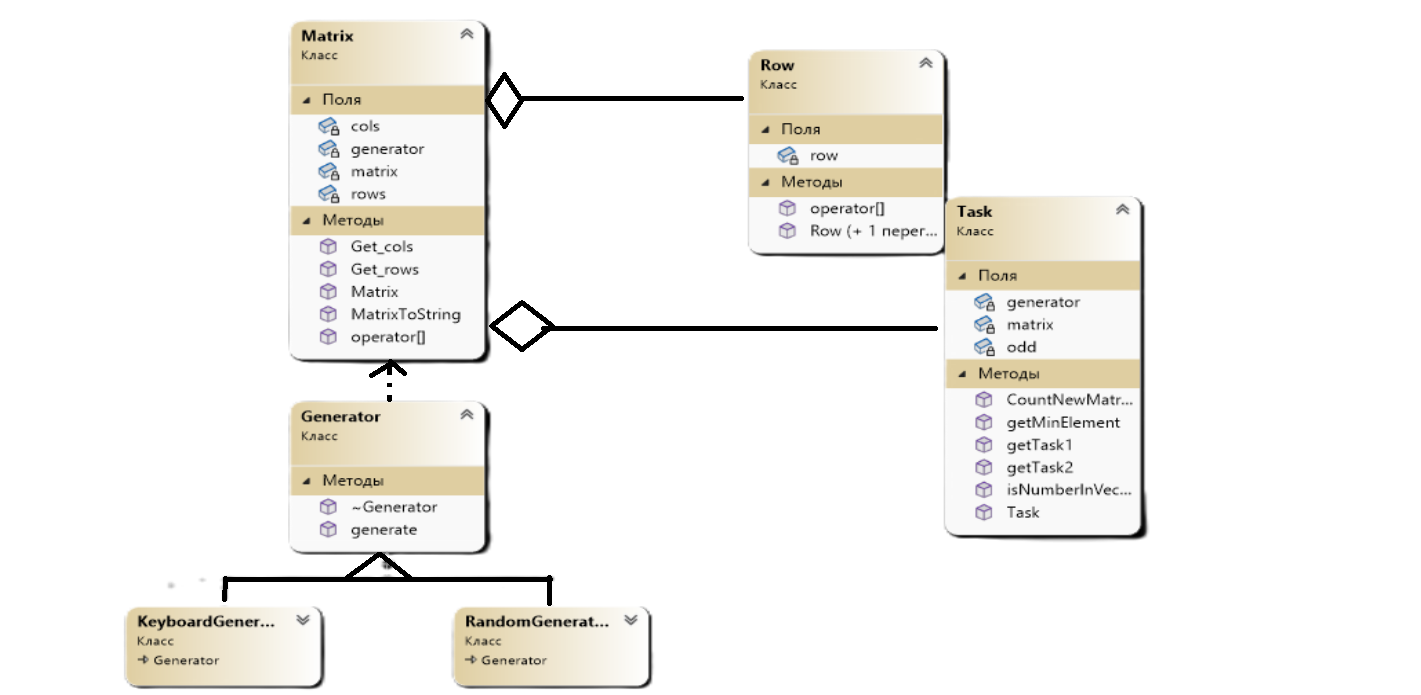


Рисунок 2 – UML диаграмма класса.

## Решение задачи на языке программирования C++

Demo.cpp:

#include <vector>

#include <string>

#include <iostream>

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

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

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

using namespace matrix;

int main() {

setlocale(LC\_ALL, "rus");

RandomGenerator rg(1, 10);

Matrix m1(2, 2, &rg);

std::cout << m1<<"\n";

Task t(m1,&rg);

Matrix m2 = t.getTask1();

std::cout << m2<<"\n";

// m1 = t.getTask1();

//std::cout << m1 << "\n";

return 1;

}

Matrix.h:

#pragma once

#include "Row.h"

#include "Generator.h"

#include <iostream>

namespace matrix {

class Matrix {

public:

//конструктор с строками, столбцами и генератором

Matrix(int rows, int cols, Generator\* generator);

std::string MatrixToString();

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

Row& operator[](const size\_t index);

int Get\_rows();

int Get\_cols();

private:

std::vector<Row>matrix;

int rows;

int cols;

Generator\* generator;

};

}

Matrix.cpp:

#include "Matrix.h"

#include <vector>

#include <string>

#include<sstream>

matrix::Matrix::Matrix(int rows, int cols, Generator\* generator):

rows(rows), cols(cols),generator(generator), matrix(rows,Row(cols))

{

if (rows <= 0 or cols <= 0 or generator==nullptr) {

throw std::out\_of\_range("Error");

}

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

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

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

}

}

}

std::string matrix::Matrix::MatrixToString()

{

std::stringstream buffer;

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

{

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

{

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

}

buffer << std::endl;

}

return buffer.str();

}

Row& matrix::Matrix::operator[](const size\_t index)

{

if (index < matrix.size()) {

return matrix[index];

}

}

int matrix::Matrix::Get\_rows()

{

return rows;

}

int matrix::Matrix::Get\_cols()

{

return cols;

}

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

{

for (const Row& row : mat.matrix) {

os << row << '\n';

}

return os;

}

Task.cpp:

#include "Task.h"

#include "Generator.h"

#include "RandomGenerator.h"

#include <cmath>

#include <vector>

Task::Task(matrix::Matrix matrix, Generator\* generator) :

matrix(matrix),generator(generator)

{

}

matrix::Matrix Task::getTask1() {

int cols = matrix.Get\_cols();

int rows = matrix.Get\_rows();

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

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

if (matrix[row][column] % 2 == 0)

{

matrix[row][column] = 0;

}

else

{

matrix[row][column] = matrix[row][column];

}

}

}

return matrix;

}

int Task::getMinElement()

{

int cols = matrix.Get\_cols();

int rows = matrix.Get\_rows();

int min = matrix[0][0];

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

{

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

{

if (matrix[row][column] < min)

{

min = matrix[row][column];

}

}

}

return min;

}

matrix::Matrix Task::getTask2()

{

bool flag = true;

size\_t rows = matrix.Get\_rows();

size\_t cols = matrix.Get\_cols();

this->odd = CountNewMatrixRows();

matrix::Matrix matrix2(rows+odd.size(), cols, generator);

rows = matrix2.Get\_rows();

cols = matrix2.Get\_cols();

int coun = 0;

flag = true;

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

for (size\_t col = 0; col < cols; col++) {

if (contains(row)) {

if (flag == 1) {

coun++;

}

matrix2[row][col] = col + 1;

flag = false;

}

else {

flag = 1;

matrix2[row][col] = matrix[row-coun][col];

}

}

}

return matrix2;

}

bool Task::contains(int number)

{

for (int i : odd) {

if (i == number) {

return true;

}

}

return false;

}

std::vector<int> Task::CountNewMatrixRows() {

bool flag = true;

size\_t rows = matrix.Get\_rows();

std::vector<int> odd;

size\_t cols = matrix.Get\_cols();

int min = getMinElement();

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

for (size\_t col = 0; col < cols; col++) {

if (matrix[row][col] == min && flag) {

odd.push\_back(row + 1);

flag = false;

}

}

flag = true;

}

return odd;

}

Generator.h:

#pragma once

#include <iostream>

#include <random>

class Generator {

public:

virtual int generate() = 0;

virtual ~Generator() = 0 {};

};

KeyboardGenerator.h:

#pragma once

#include"Generate.h"

class KeyboardGenerator : public Generator {

private:

std::istream& in;

public:

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

int generate() override;

};

KeyboardGenerator.cpp:

#include "KeyboardGenerator.h"

KeyboardGenerator::KeyboardGenerator(std::istream& in):

in(in)

{

}

int KeyboardGenerator::generate()

{

int value = 0;

this->in >> value;

return value;

}

RandomGenerator.h:

#pragma once

#include <random>

#include"Generate.h"

class RandomGenerator : public Generator {

private:

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

std::mt19937 generator;

public:

RandomGenerator(const int min, const int max);

// Реализация функции генерации случайного числа

int generate() override;

};

RandomGenerator.cpp:

#include "RandomGenerator.h"

RandomGenerator::RandomGenerator(const int min, const 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);

}

## Решение тестов

Код тестов:

#include "pch.h"

#include "CppUnitTest.h"

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

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

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

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

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

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

using namespace matrix;

namespace MatrixTest

{

TEST\_CLASS(MatrixTest)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

RandomGenerator gen(-10, 10);

Matrix m(3, 3, &gen);

Assert::IsNotNull(&m);

}

TEST\_METHOD(Copy\_Test\_Success) {

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 3, &rnd);

Matrix matrixtwo(matrix);

Assert::AreEqual(matrix.MatrixToString(), matrixtwo.MatrixToString());

}

TEST\_METHOD(GetCol\_Test\_Validdata\_Succes) {

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 5, &rnd);

int expcol = 5;

Assert::AreEqual(matrix.Get\_cols(), expcol);

}

TEST\_METHOD(GetRow\_Test\_Validdata\_Succes) {

RandomGenerator rnd(-10, 10);

Matrix matrix(3, 5, &rnd);

int expcol = 3;

Assert::AreEqual(matrix.Get\_rows(), expcol);

}

};

TEST\_CLASS(TaskTest)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

RandomGenerator rnd(-5, 5);

Matrix matrix(3, 3, &rnd);

Task task(matrix,&rnd);

Assert::IsNotNull(&task);

}

TEST\_METHOD(GetTask2\_ValidData\_Success)

{

std::istringstream in("1 2 3 5");

KeyboardGenerator kbd(in);

Matrix matrix(2, 2, &kbd);

Task task(matrix, &kbd);

Matrix matrix2 = task.getTask2();

std::string exp = "1 2 \n1 2 \n3 5 \n";

bool AreEqual = (exp == matrix2.MatrixToString());

Assert::IsTrue(AreEqual);

}

TEST\_METHOD(GetTask1\_ValidData\_Success)

{

std::istringstream in("1 2 3 5");

KeyboardGenerator kbd(in);

Matrix matrix(2, 2, &kbd);

Task task(matrix, &kbd);

Matrix matrix2 = task.getTask1();

std::string exp = "1 0 \n3 5 \n";

bool AreEqual = (exp == matrix2.MatrixToString());

Assert::IsTrue(AreEqual);

}

};

}

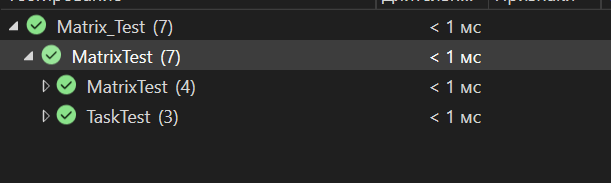


Рисунок 4 – Успешные тесты.