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 № 3

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Борцов Ю. Д.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Задание:** Написать класс матрица с реализацией задач из задания 4.3

**Код программы:**

#include "Exersize.h"

#include "RandomGenerator.h"

Exersize::Exersize(Matrix matrix) : matrix(matrix) {}

Matrix Exersize::Exersize1()

{

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

{

size\_t max = matrix[i][0];

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

{

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

{

max = matrix[i][j];

}

}

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

{

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

{

matrix[i][j] = 0;

}

}

}

return matrix;

}

Matrix Exersize::Exersize2()

{

size\_t newRows = matrix.getRows();

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

{

if (matrix[i][0] % 3 == 0)

{

newRows += 1;

}

}

RandomGenerator rnd(0, 0);

Matrix newMatrix(newRows, matrix.getColumns(), &rnd);

size\_t newIndex = 0;

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

{

if (matrix[i][0] % 3 == 0)

{

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

{

newMatrix[newIndex][j] = 0;

}

newIndex += 1;

}

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

{

newMatrix[newIndex][j] = matrix[i][j];

}

newIndex += 1;

}

return newMatrix;

}

#pragma once

#include "Exersizes.h"

/\*\*

\* @brief класс с заданиями 1 и 2.

\*/

class Exersize : public ExersizeAbs

{

private:

Matrix matrix;

public:

/\*\*

\* @brief конструктор

\*/

Exersize(Matrix matrix);

/\*\*

\* @brief метод для задания 1.

\*/

Matrix Exersize1() override;

/\*\*

\* @brief метод для задания 2.

\*/

Matrix Exersize2() override;

};

#include "Exersizes.h"

ExersizeAbs::~ExersizeAbs() {}

#pragma once

#include "Matrix.h"

#include "Generator.h"

/\*\*

\* @brief Абстрактный класс для заданий

\*/

class ExersizeAbs

{

public:

/\*\*

\* @brief Виртуальный деструтор.

\*/

virtual ~ExersizeAbs() = 0;

/\*\*

\* @brief Виртуальный метод для первого задания.

\*/

virtual Matrix Exersize1() = 0;

/\*\*

\* @brief Виртуальный метод для второго задания

\*/

virtual Matrix Exersize2() = 0;

};

#include "Generator.h"

Generator::~Generator() {}

#pragma once

/\*\*

\* \brief Абстрактный класс для генератора случайных чисел.

\*/

class Generator {

public:

/\*\*

\* \brief Виртуальный деструктор.

\*/

virtual ~Generator() = 0;

/\*\*

\* \brief Виртуальный метод для генерации случайного числа.

\*/

virtual int generate() = 0;

};

#include "IStreamGenerator.h"

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

{

}

int IStreamGenerator::generate()

{

int value = 0;

this->in >> value;

return value;

}

#pragma once

#include "Generator.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;

};

#include "Matrix.h"

#include <iostream>

#include <vector>

#include <sstream>

Matrix::Matrix(const int rows, const int columns, Generator\* generator) : rows(rows), columns(columns), matrix(rows, std::vector<int>(columns))

{

if (generator && columns && rows > 0)

{

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

{

for (size\_t j = 0; j < static\_cast<size\_t>(columns); ++j)

{

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

}

}

}

}

std::string Matrix::matrixString()

{

std::stringstream buffer;

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

{

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

{

buffer << matrix[i][j] << '\t';

}

buffer << "\n";

}

return buffer.str();

}

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

{

os << matrix.matrixString();

return os;

}

size\_t Matrix::getColumns()

{

return columns;

}

size\_t Matrix::getRows()

{

return rows;

}

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

{

if (index < matrix.size())

{

return matrix[index];

}

else

{

std::cerr << "Ошибка: индекс за пределами вектора.";

static std::vector<int> dummy;

return dummy;

}

}

std::vector<std::vector<int>> Matrix::getMatrix()

{

return this->matrix;

}#include "Generator.h"

#include <iostream>

#include <vector>

/\*\*

\* \brief Класс для представления матрицы целых чисел.

\*/

class Matrix

{

private:

int rows;

int columns;

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

public:

/\*\*

\* \brief Конструктор с заданным размером матрицы.

\* \param rows Количество строк.

\* \param columns Количество столбцов.

\*/

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

/\*\*

\* \brief Деструктор.

\*/

~Matrix() = 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 matrixString();

/\*\*

\* \brief Метод для получения количества столбцов.

\*/

size\_t getColumns();

/\*\*

\* \brief Метод для получения количества строк.

\*/

size\_t getRows();

/\*\*

\* \brief Конструктор копирования.

\* \param other Ссылка на другой объект Matrix для копирования.

\*/

Matrix(const Matrix& other) = default;

/\*\*

\* \brief Оператор присваивания.

\* \param other Ссылка на другой объект Matrix для присваивания.

\*/

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

/\*\*

\* \brief Конструктор перемещения.

\* \param other R-value ссылка на другой объект Matrix для перемещения.

\*/

Matrix(Matrix&& other) noexcept = default;

/\*\*

\* \brief Оператор перемещения.

\* \param other R-value ссылка на другой объект Matrix для перемещения.

\*/

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

/\*\*

\* \brief Получение элемента матрицы.

\* \param rows Номер строки.

\* \param columns Номер столбца.

\*/

int getElement(int rows, int columns);

/\*\*

\* \brief Переопределение оператора []

\* \param index Индекс матрицы.

\*/

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

/\*\*

\* @brief Получение матрицы

\*/

std::vector<std::vector<int>> getMatrix();

};

#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);

}

#pragma once

#include "Generator.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;

};

**Tecты:**

#include "gtest/gtest.h"

#include "Exersize.h"

#include "RandomGenerator.h"

#include "IStreamGenerator.h"

#include <sstream>

TEST(Exersize1, Exersize1)

{

Matrix matrix(2, 2, new RandomGenerator(1, 1));

Exersize exersize(matrix);

Matrix result = exersize.Exersize1();

EXPECT\_EQ(result.matrixString(), "0\t0\t\n0\t0\t\n");

}

TEST(Exersize2, Exersize2)

{

Matrix matrix(2, 2, new RandomGenerator(3, 3));

Exersize exersize(matrix);

Matrix result = exersize.Exersize2();

EXPECT\_EQ(result.matrixString(), "0\t0\t\n3\t3\t\n0\t0\t\n3\t3\t\n");

}

TEST(MatrixTest, Constructor)

{

Matrix matrix1(2, 2, new RandomGenerator(1, 1));

Matrix matrix2(3, 3, new RandomGenerator(1, 1));

EXPECT\_EQ(matrix1.getRows(), 2);

EXPECT\_EQ(matrix1.getColumns(), 2);

EXPECT\_EQ(matrix2.getRows(), 3);

EXPECT\_EQ(matrix2.getColumns(), 3);

}

TEST(MatrixTest, MatrixString)

{

Matrix matrix(2, 2, new RandomGenerator(1, 1));

std::string matrixStr = matrix.matrixString();

EXPECT\_EQ(matrixStr, "1\t1\t\n1\t1\t\n");

}

TEST(MatrixTest, OperatorBracket)

{

Matrix matrix(2, 2, new RandomGenerator(1, 10));

std::vector<int>& row = matrix[0];

row[0] = 42;

EXPECT\_EQ(matrix[0][0], 42);

}

TEST(MatrixTest, OperatorOutput)

{

Matrix matrix(2, 2, new RandomGenerator(1, 1));

std::stringstream ss;

ss << matrix;

std::string matrixStr = ss.str();

EXPECT\_EQ(matrixStr, "1\t1\t\n1\t1\t\n");

}

TEST(RandomGeneratorTest, Generate)

{

RandomGenerator generator(1, 10);

int value = generator.generate();

EXPECT\_GE(value, 1);

EXPECT\_LE(value, 10);

}

TEST(IStreamGeneratorTest, Generate)

{

std::istringstream input("42");

IStreamGenerator streamGenerator(input);

int value = streamGenerator.generate();

EXPECT\_EQ(value, 42);

}

TEST(RandomGeneratorTest, GenerateRange)

{

RandomGenerator generator(5, 10);

int value = generator.generate();

EXPECT\_GE(value, 5);

EXPECT\_LE(value, 10);

}

TEST(RandomGeneratorTest, GenerateMultiple)

{

RandomGenerator generator(1, 1);

EXPECT\_EQ(generator.generate(), generator.generate());

EXPECT\_EQ(generator.generate(), generator.generate());

}

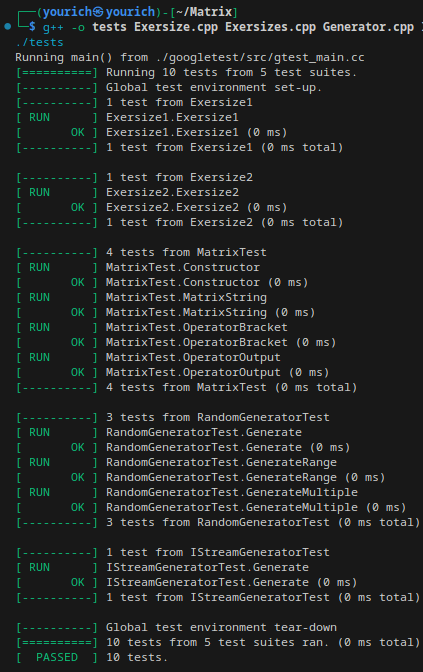


Рисунок 1 – прохождение тестов

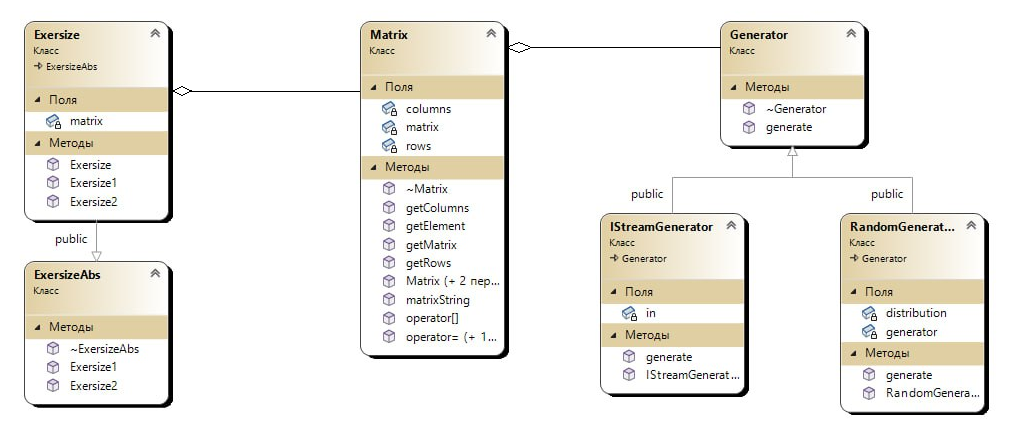
****

Рисунок 2 – UML-диаграмма проекта

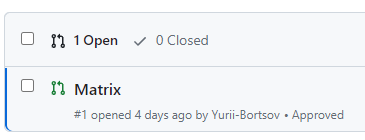


Рисунок 3 – Approve задания