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

Дьячков Денис Андреевич

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Формулировка задания:**

Написать матрицу

**Код:**

**Generator:**

#pragma once

/\*\*

\* @brief Базовый класс типа генератор

\*/

class Generator

{

public:

/\*\*

\* @brief Чисто виртуальный деструктор для удаления памяти за наследниками

\*/

virtual ~Generator() = 0 {};

/\*\*

\* @brief Чисто виртуальный генератор чисел у наследников

\*/

virtual int generate() = 0;

};

**Matrix:**

#pragma once

#include <iostream>

#include <vector>

#include "Generator.h"

namespace miit

{

class Matrix

{

private:

std::vector<std::vector<int>> info;

public:

/\*\*

\* @brief Создает объект матрицы

\* @param gen Генератор для заполнения

\* @param rows Количество строк

\* @param columns Количество столбцов

\*/

Matrix(Generator\* gen, size\_t rows, size\_t columns);

/\*\*

\* @brief Создает объект матрицы

\* @param info Информация о данных

\* @param rows Количество строк

\* @param columns Количество столбцов

\*/

Matrix(std::initializer\_list<int> info, size\_t rows, size\_t columns);

/\*\*

\* @brief Создает пустой объект матрицы

\*/

Matrix();

/\*\*

\* @brief Добавляет в конец матрицы строку

\* @param element Строка, которая будет добавлена

\*/

void append\_row(std::vector<int> element);

/\*\*

\* @brief Узнает сколько строк в матрице

\* @return Количество строк в матрице

\*/

size\_t get\_rows() const;

/\*\*

\* @brief Узнает сколько столбцов в матрице

\* @return Количество столбцов в матрице

\*/

size\_t get\_columns() const;

/\*\*

\* @brief Узнает минимальное значение в матрице

\* @return Минимальное значение в матрице

\*/

int get\_min\_value() const;

/\*\*

\* @brief Узнает есть ли элемент в строке

\* @param index Индекс строки

\* @param element Элемент доя поиска

\* @return если есть true, иначе false

\*/

bool is\_value\_in\_row(size\_t index, int element);

/\*\*

\* @brief Перегрузка оператора вывода

\* @param os Поток вывода

\* @param matrix Матрица для вывода

\* @return Поток вывода с матрицей

\*/

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

/\*\*

\* @brief Перегрузка оператора сравнения двух матриц

\* @param lhs Первая матрица для сравнения

\* @param rhs Вторая матрица для сравнения

\* @return если равны true, иначе false

\*/

friend bool operator==(Matrix& lhs, Matrix& rhs) noexcept;

/\*\*

\* @brief Перегрузка оператора индекса

\* @param index Индекс элемента в матрице

\* @return Элемент под этим индексом

\*/

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

/\*\*

\* @brief Перегрузка оператора индекса

\* @param index Индекс элемента в матрице

\* @return Элемент под этим индексом

\*/

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

/\*\*

\* @brief Преобразование матрицы в строку

\* @return Строка с матрицей

\*/

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

};

}

#include "Header.h"

#include <sstream>

miit::Matrix::Matrix(Generator\* gen, size\_t rows, size\_t columns)

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

{

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

{

std::vector<int> row{};

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

{

row.push\_back(gen->generate());

}

this->info.push\_back(row);

}

}

miit::Matrix::Matrix(std::initializer\_list<int> info, size\_t rows, size\_t columns)

:info(std::vector<std::vector<int>> {rows})

{

size\_t i = 0;

size\_t j = 0;

for (int item : info)

{

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

i++;

if (i % columns == 0)

{

j++;

}

}

}

miit::Matrix::Matrix()

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

{

}

void miit::Matrix::append\_row(std::vector<int> element)

{

this->info.push\_back(element);

}

size\_t miit::Matrix::get\_rows() const

{

return info.size();

}

size\_t miit::Matrix::get\_columns() const

{

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

{

return 0;

}

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

}

int miit::Matrix::get\_min\_value() const

{

int min = std::numeric\_limits<int>::max();

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

{

for (size\_t j = 0; j < this->get\_columns(); j++)

{

if (this->info[i][j] <= min)

{

min = info[i][j];

}

}

}

return min;

}

bool miit::Matrix::is\_value\_in\_row(size\_t index, int element)

{

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

{

if (this->info[index][i] == element)

{

return true;

}

}

return false;

}

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

{

return info[index];

}

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

{

return info[index];

}

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

{

std::stringstream buffer;

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

{

for (size\_t j = 0; j < this->get\_columns(); j++)

{

buffer << this->info[i][j] << " ";

}

buffer << "\n";

}

return buffer.str();

}

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

{

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

}

bool miit::operator==(Matrix& lhs, Matrix& rhs) noexcept

{

return (lhs.to\_string() == rhs.to\_string());

}

**IStreamGenerator:**

#pragma once

#include "Generator.h"

#include <iostream>

/\*\*

\* @brief Класс генератор для ввода чисел с клавиатуры

\*/

class IStreamGen : public Generator

{

private:

std::istream& in;

public:

/\*\*

\* @brief Инициализация объекта типа IStreamGenerator

\* @param input Поток ввода

\*/

IStreamGen(std::istream& = std::cin);

/\*\*

\* @brief Запрашивает у пользователся число

\* @return Элемент от пользователя

\*/

int generate() override;

};

#include "IStreamGen.h"

IStreamGen::IStreamGen(std::istream& in)

:in{in}

{

}

int IStreamGen::generate()

{

int value = 0;

this->in >> value;

return value;

}

**RandomGenerator:**

#pragma once

#include <random>

#include "Generator.h"

/\*\*

\* @brief Класс генератора случайных чисел

\*/

class RandomGen : public Generator

{

private:

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

std::mt19937 generator;

public:

/\*\*

\* @brief Создание объекта типа RandomGen

\* @param left Левая граница диапазона

\* @param right Правая граница диапазона

\*/

RandomGen(int left, int right);

/\*\*

\* @brief Создает случайное число из промежутка

\* @return Случайное число из промежутка

\*/

int generate() override;

};

#include "RandomGen.h"

RandomGen::RandomGen(int left, int right)

{

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

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

}

int RandomGen::generate()

{

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

}

**Код тестов:**

#include "pch.h"

#include "CppUnitTest.h"

#include "../MatrixLib/Header.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

using namespace miit;

namespace Tests

{

TEST\_CLASS(Tests)

{

public:

TEST\_METHOD(MatrixAppendRow\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Matrix matrix\_1 = Matrix({ 1,2,3,4,5,6 }, 3, 2);

std::vector<int> row{ 5, 6 };

matrix.append\_row(row);

Assert::IsTrue(matrix\_1 == matrix);

}

TEST\_METHOD(MatrixToString\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Assert::IsTrue(matrix.to\_string() == "1 2 \n3 4 \n");

}

TEST\_METHOD(MatrixEqualOperator\_ValidData\_Success)

{

Matrix matrix\_1 = Matrix({ 1,2,3,4 }, 2, 2);

Matrix matrix\_2 = Matrix({ 1,2,3,4,5,6 }, 3, 2);

Matrix matrix\_3 = Matrix({ 1,2,3,4 }, 2, 2);

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

}

TEST\_METHOD(MatrixIndexOperator\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

int element = 4;

Assert::IsTrue(matrix[1][1] == element);

}

TEST\_METHOD(MatrixGetRows\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Assert::IsTrue(matrix.get\_rows() == 2);

}

TEST\_METHOD(MatrixRows\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Assert::IsTrue(matrix.get\_columns() == 2);

}

TEST\_METHOD(MatrixGetMinValue\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Assert::IsTrue(matrix.get\_min\_value() == 1);

}

TEST\_METHOD(MatrixIsValueInRow\_ValidData\_Success)

{

Matrix matrix = Matrix({ 1,2,3,4 }, 2, 2);

Assert::IsTrue(matrix.is\_value\_in\_row(1, 3) == true);

}

};

}

**Прохождение тестов:**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание**

Рисунок 1 - Пройденные тесты

**UML диаграмма кода:**

**Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание**

Рисунок 2 - UML диаграмма кода

**Approve:**

Рисунок 3 - Approve