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

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Владимиров М. А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2024

Содержание

1. Формулировка задачи

2. UML диаграмма

3. Решение задачи на языке программирования C++

4. Тесты

5. Зачет задания в GitHub

**Задание 4–1**

1. **Формулировка задачи**

Написать матрицу в ООП.

1. **UML диаграмма**

**Изображение выглядит как текст, диаграмма, линия, снимок экрана

Автоматически созданное описание**

1. **Решение задачи на языке программирования C++**

﻿**Matrix.h**

#pragma once

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

#include <iostream>

#include <vector>

#include <sstream>

class Matrix

{

private:

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

public:

/\*\*

\* @brief Конструктор матрицы.

\* @param Numrows Число строк.

\* @param Numcols Число столбцов.

\* @exception Если заданы неверные параметры.

\*/

Matrix(size\_t numrows, size\_t numcols, Generator\* generator);

/\*\*

\* @brief Получить количество строк.

\* @return Количество строк.

\*/

size\_t GetRows();

/\*\*

\* @brief Получить количество столбцов.

\* @return Количество столбцов.

\*/

size\_t GetCols();

/\*\*

\* @brief Функция записывающая матрицу в строку.

\* @return Строка из матрицы.

\*/

std::string ToString();

/\*\*

\* @brief Перегрузка оператора квадратных скобок.

\* @param index Индекс.

\* @exception Если индекс неправильный.

\* @return Элемент по индексу.

\*/

int& operator[](std::pair<int,int> index);

/\*\*

\* @brief Перегрузка оператора <<.

\* @param os Поток вывода.

\* @param matrix матрица.

\* @return os Возвращает матрицу в поток вывода.

\*/

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

/\*\*

\* @brief Assignment operator.

\* @param other Other matrix.

\* @return \*this.

\*/

Matrix& operator=(Matrix other);

/\*\*

\* @brief Swap function.

\* @param other Other matrix.

\*/

void swap(Matrix& other);

/\*\*

\* @brief Оператор сравнения.

\* @param second Матрица с которой сравниваем.

\* @return true Если матрицы равны.

\*/

bool operator==(Matrix& second);

};

**Matrix.cpp**

﻿#include "Matrix.h"

Matrix::Matrix(size\_t numrows, size\_t numcols, Generator\* generator) : matrix({})

{

if (numrows < 0)

{

throw std::out\_of\_range("Wrong rows count");

}

if (numcols < 0)

{

throw std::out\_of\_range("Wrong columns count");

}

if (generator == nullptr)

{

throw std::invalid\_argument("Wrong generator");

}

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

{

std::vector<int> temp{};

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

{

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

}

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

}

}

size\_t Matrix::GetRows()

{

return this->matrix.size();

}

size\_t Matrix::GetCols()

{

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

}

std::string Matrix::ToString()

{

std::stringstream buffer;

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

{

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

{

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

}

buffer << std::endl;

}

return buffer.str();

}

int& Matrix::operator[](std::pair<int,int> index)

{

size\_t numRows = index.first;

size\_t numCols = index.second;

if (numRows >= this->GetRows() || numCols >= this->GetCols())

{

throw std::out\_of\_range("Invalid Index");

}

return matrix[numRows][numCols];

}

Matrix& Matrix::operator=(Matrix other)

{

swap(other);

return \*this;

}

void Matrix::swap(Matrix& other)

{

std::swap(this->matrix, other.matrix);

}

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

{

for (int i = 0; i < matrix.GetRows(); i++)

{

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

{

os << matrix[{i,j}] << " ";

}

os << std::endl;

}

return os;

}

bool Matrix::operator==(Matrix& second)

{

return (this->ToString() == second.ToString());

}

**Random.h**

﻿#pragma once

#include "Generator.h"

#include <random>

class Random : public Generator

{

private:

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

std::mt19937 generator;

public:

/\*\*

\* @brief Случайная генерация.

\* @param min Минимальное значение.

\* @param max Максимальное значение.

\*/

Random(const int min, const int max);

/\*\*

\* @brief Генерация числа.

\* @return Число.

\*/

int generate() override;

};

**Random.cpp**

﻿#include "Random.h"

Random::Random(const int min, const int max)

{

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

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

}

int Random::generate()

{

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

}

**IStream.h**

﻿#pragma once

#include "Generator.h"

#include <iostream>

class Istream : public Generator

{

private:

std::istream& in;

public:

/\*\*

\* @brief Создание потока ввода.

\* @param in поток ввода.

\*/

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

/\*\*

\* @brief

\*/

int generate() override;

};

**IStream.cpp**

﻿#include "IStream.h"

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

{

}

int Istream::generate()

{

int value = 0;

this->in >> value;

return value;

}

**Generator.h**

﻿#pragma once

class Generator

{

public:

/\*\*

\* @brief Виртуальный деструктор.

\*/

virtual ~Generator() = 0 {};

/\*\*

\* @brief Генерация числа.

\*/

virtual int generate() = 0;

};

**ConstantGenerator.h**

﻿#pragma once

#include "Generator.h"

#include <iostream>

class ConstGenerator: public Generator

{

private:

int value = 0;

public:

/\*\*

\* @brief Создание потока ввода.

\* @param in поток ввода.

\*/

ConstGenerator(int value);

/\*\*

\* @brief

\*/

int generate() override;

};

**ConstantGenerator.cpp**

#include "ConstantGenerator.h"

ConstGenerator::ConstGenerator(int value) : value(value)

{

}

int ConstGenerator::generate()

{

return value;

}

**Main.cpp**

﻿#include <iostream>

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

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

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

#include "../Matrix11/Matrix.h"

#include "../Task1/Task.h"

int main()

{

setlocale(LC\_ALL, "Rus");

try

{

Generator\* constant = new ConstGenerator(2);

Matrix A(2, 2, constant);

//Matrix B(2, 2, constant);

//Task Task1{A};

//std::cout << Task1.Subtract(A,B).ToString();

std::cout << A[{0,100}];

return 0;

}

catch (std::exception& e)

{

std::cerr << e.what();

return 1;

}

}

**Task.h**

#pragma once

#include "../Matrix11/Matrix.h"

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

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

class Task

{

public:

/\*\*

\* @brief Конструктор.

\*/

Task(Matrix matrix);

/\*

\* @brief Функция замеяющая нулём минимальный элемент каждой строки.

\* @return Матрица с нулями.

\*/

Matrix set\_min\_zero();

/\*\*

\* @brief Сложение двух матриц.

\* @param lha Первая матрица.

\* @param rha Вторая матрица.

\* @exception Если размеры матриц не равны.

\* @return Sum.

\*/

Matrix Sum(Matrix& lha, Matrix& rha);

/\*\*

\* @brief Вычитание двух матриц.

\* @param lha Первая матрица.

\* @param rha Вторая матрица.

\* @exception Если размеры матриц не равны.

\* @return result.

\*/

Matrix Subtract(Matrix& lha, Matrix& rha);

/\*\*

\* @brief Умножение двух матриц.

\* @param lha Первая матрица.

\* @param rha Вторая матрица.

\* @exception Если матрицы не соответствуют условию умножения.

\* @return result.

\*/

Matrix Multiply(Matrix& lha, Matrix& rha);

private:

Matrix matrix;

};

**Task.cpp**

﻿#include "Task.h"

Task::Task(Matrix matrix) : matrix(matrix)

{

}

Matrix Task::set\_min\_zero()

{

Matrix temp{ this->matrix };

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

{

int minelem = temp[{i,0}];

for (size\_t j = 1; j < this->matrix.GetCols(); j++)

{

if (matrix[{i,j}] < minelem)

{

minelem = temp[{i,j}];

}

}

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

{

if (temp[{i,j}] == minelem)

{

temp[{i,j}] = 0;

}

}

}

return temp;

}

Matrix Task::Sum(Matrix& lha, Matrix& rha)

{

if (lha.GetCols() != rha.GetCols() || lha.GetRows() != rha.GetRows())

{

throw std::out\_of\_range("Different matrix sizes");

}

Matrix sum = lha;

for (size\_t i = 0; i < lha.GetRows(); i++)

{

for (size\_t j = 0; j < lha.GetCols(); j++)

{

sum[{i,j}] = sum[{i, j}] + rha[{i, j}];

}

}

return sum;

}

Matrix Task::Subtract(Matrix& lha, Matrix& rha)

{

if (lha.GetCols() != rha.GetCols() || lha.GetRows() != rha.GetRows())

{

throw std::out\_of\_range("Different matrix sizes");

}

Matrix result = lha;

for (size\_t i = 0; i < lha.GetRows(); i++)

{

for (size\_t j = 0; j < lha.GetCols(); j++)

{

result[{i,j}] = result[{i,j}] - rha[{i,j}];

}

}

return result;

}

Matrix Task::Multiply(Matrix& lha, Matrix& rha)

{

size\_t newRows = rha.GetRows();

size\_t newCols = lha.GetCols();

if (newRows != newCols)

{

throw std::out\_of\_range("Число строк матрицы 2 не равно числу столбцов матрицы 1");

}

Generator\* temp = new ConstGenerator(0);

Matrix result(newRows, newCols, temp);

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

{

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

{

for (size\_t k = 0; k < lha.GetCols(); k++)

{

result[{i,j}] += this->matrix[{i,k}] \* rha[{k,j}];

}

}

}

return result;

}

**UnitTest1.cpp**

﻿#include "pch.h"

#include "CppUnitTest.h"

#include "../Matrix11/Matrix.h"

#include "../Task1/Task.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1

{

TEST\_CLASS(UnitTest1)

{

public:

TEST\_METHOD(ToString\_ValidData\_Success)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 4, gen);

Assert::IsTrue(A.ToString() == "2 2 2 2 \n2 2 2 2 \n");

}

TEST\_METHOD(GetCols\_ValidData\_Success)

{

Generator\* constant = new ConstGenerator(2);

Matrix A(2, 2, constant);

Assert::IsTrue(A.GetCols()==2);

}

TEST\_METHOD(GetRows\_ValidData\_Success)

{

Generator\* constant = new ConstGenerator(2);

Matrix A(3, 2, constant);

Assert::IsTrue(A.GetRows() == 3);

}

TEST\_METHOD(Sum\_ValidData\_Success)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(2, 2, gen);

Generator\* gen2 = new ConstGenerator(4);

Matrix C(2, 2, gen2);

Task task\_1{ A };

Assert::IsTrue(task\_1.Sum(A,B).ToString() == C.ToString());

}

TEST\_METHOD(Subtract\_ValidData\_Success)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Generator\* gen1 = new ConstGenerator(5);

Matrix B(2, 2, gen1);

Generator\* gen2 = new ConstGenerator(-3);

Matrix C(2, 2, gen2);

Task task\_1{ A };

Assert::IsTrue(task\_1.Subtract(A, B).ToString() == C.ToString());

}

TEST\_METHOD(Sum\_InvalidSizes\_ExpectedException)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(2, 3, gen);

Task task\_1 { A };

Assert::ExpectException<std::out\_of\_range>([&]()

{ task\_1.Sum(A, B); });

}

TEST\_METHOD(Sum\_InvalidSizes\_ExpectedException2)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(3, 2, gen);

Task task\_1{ A };

Assert::ExpectException<std::out\_of\_range>([&]()

{ task\_1.Sum(A, B); });

}

TEST\_METHOD(Sum\_InvalidSizes\_ExpectedException3)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(3, 3, gen);

Task task\_1{ A };

Assert::ExpectException<std::out\_of\_range>([&]()

{ task\_1.Sum(A, B); });

}

TEST\_METHOD(Subtract\_InvalidSizes\_ExpectedException)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(3, 2, gen);

Task task\_1{ A };

Assert::ExpectException<std::out\_of\_range>([&]()

{ task\_1.Subtract(A, B); });

}

TEST\_METHOD(Multiply\_InvalidSizes\_ExpectedException)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Matrix B(3, 2, gen);

Task task\_1{ A };

Assert::ExpectException<std::out\_of\_range>([&]()

{ task\_1.Multiply(A, B); });

}

TEST\_METHOD(IndexOperator\_InvalidIndex\_ExpectedException)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Assert::ExpectException<std::out\_of\_range>([&]()

{ std::cout << A[{-200,0}]; });

}

TEST\_METHOD(IndexOperator\_InvalidIndex\_ExpectedException2)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Assert::ExpectException<std::out\_of\_range>([&]()

{ std::cout << A[{0,100}]; });

}

TEST\_METHOD(IndexOperator\_InvalidIndex\_ExpectedException3)

{

Generator\* gen = new ConstGenerator(2);

Matrix A(2, 2, gen);

Assert::ExpectException<std::out\_of\_range>([&]()

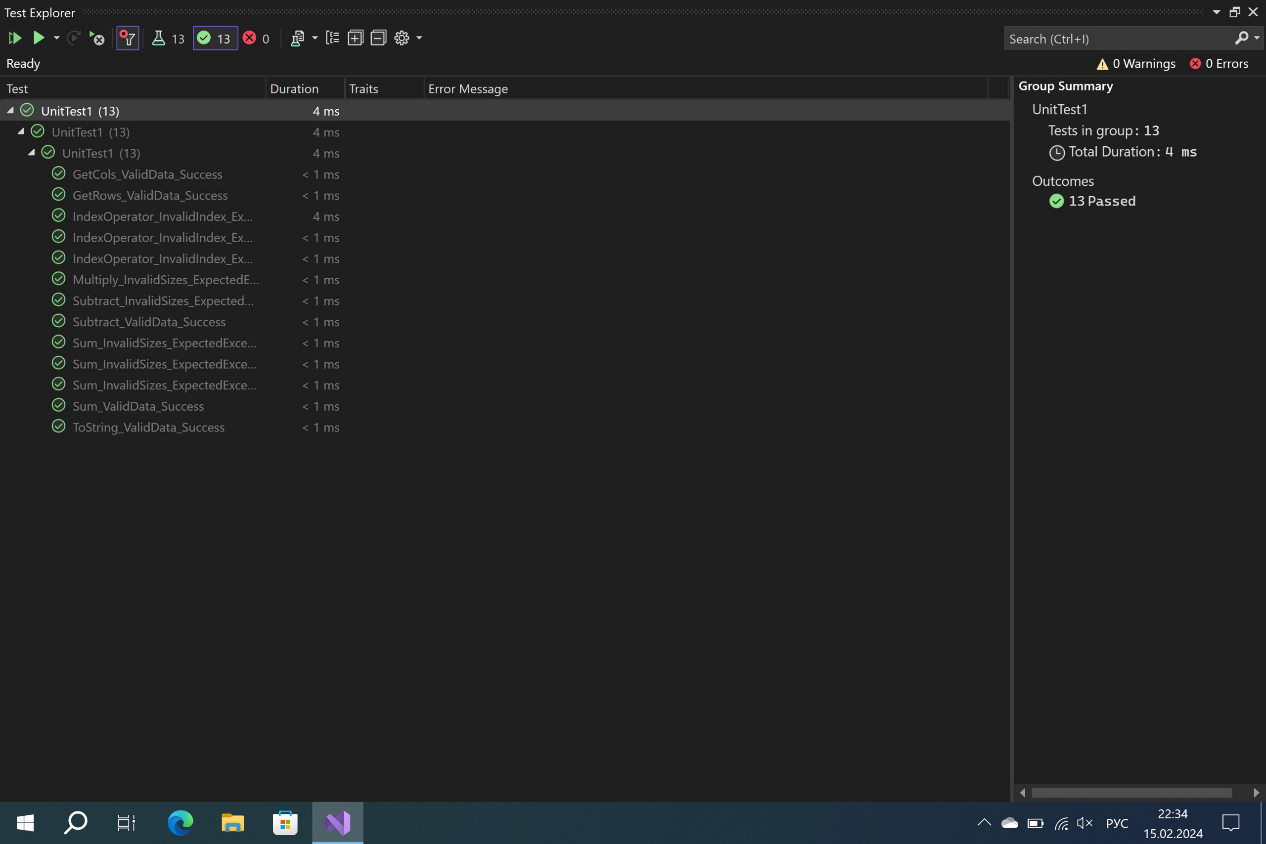
{ std::cout << A[{11,-4}]; });

}

};

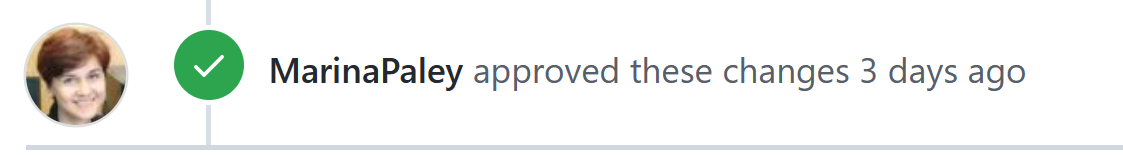
}

1. **Тесты**

****

**Рисунок - Тесты**

1. **Зачет задания в GitHub**



**Рисунок – Зачет задания в GitHub**