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

Байло А.В.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

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

**Код программы:**

#include "row.h"

#include <initializer\_list>

#include <sstream>

#include <string>

#include <vector>

/\*

@brief Class Matrix

\*/

class Matrix {

public:

Matrix(std::initializer\_list<int> list, int rows, int columns);

/\*

@brief Override of [] operator

@param index of necessary element

@return value of element (row)

\*/

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

/\*

@brief Override of << operator

@param os output stream

@param matrix object to be put in stream

@return output stream contains matrix

\*/

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

/\*

@brief function which transpose a matrix

\*/

void transpose();

/\*

@brief function which delete column in matrix

@param index of deleted colunm

\*/

void deleteColumn(size\_t index);

/\*

@brief function which delete row in matrix

@param index of deleted row

\*/

void deleteRow(size\_t index);

/\*

@brief function which convert matrix into std::string

@return string which contains matrix

\*/

std::string toString();

size\_t getSize() const;

const std::vector<Row> getMatrix() const;

private:

std::vector<Row> matrix;

};

#include "matrix.h"

#include <iostream>

#include <stdexcept>

#include <algorithm>

#include <vector>

Matrix::Matrix(std::initializer\_list<int> list, int rows, int columns)

{

std::vector<int> list1;

for (auto& i: list)

{

list1.push\_back(i);

}

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

{

Row initRow {};

matrix.push\_back(initRow);

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

{

matrix[i].row.push\_back(list1[columns\*i + j]);

}

}

}

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

{

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

{

for (size\_t j = 0; j < matrix.matrix[i].row.size(); j++)

{

os << matrix.matrix[i].row[j] << ' ';

}

os << '\n';

}

return os;

}

void Matrix::transpose()

{

std::vector<Row> newMatrix;

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

{

Row newRow{};

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

{

newRow.row.push\_back(this->matrix[i][j]);

}

newMatrix.push\_back(newRow);

}

this->matrix = newMatrix;

}

void Matrix::deleteColumn(size\_t index)

{

this->transpose();

deleteRow(index);

this->transpose();

}

void Matrix::deleteRow(size\_t index)

{

matrix.erase(matrix.begin() + index);

}

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

{

return matrix[index];

}

std::string Matrix::toString()

{

std::stringstream buffer;

for(auto it = matrix.begin(); it != matrix.end(); it++)

{

for (auto subIt = it->row.begin(); subIt != it->row.end(); subIt++)

{

buffer << \*subIt;

}

}

return buffer.str();

}

size\_t Matrix::getSize() const

{

return this->matrix.size();

}

const std::vector<Row> Matrix::getMatrix() const

{

return this->matrix;

}

#include <initializer\_list>

#include <iostream>

#include <vector>

/\*

@brief Struct row

\*/

struct Row

{

/\*

@brief override [] operator

@param index of necessary element

@return value of element

\*/

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

std::vector<int> row;

/\*

@brief function which searching minimal element in row

\*/

int searchMinimun();

};

#include "row.h"

#include <algorithm>

#include <cstddef>

#include <vector>

int Row::searchMinimun()

{

std::vector<int>::iterator it = std::min\_element(row.begin(), row.end());

return \*it;

}

int& Row::operator[](size\_t index)

{

return row[index];

}

#pragma once

#include "matrix.h"

struct BasicTask

{

public:

/\*\*

\* @brief Чисто виртуальный метод для решения первого задания

\*/

virtual Matrix task1() = 0;

/\*\*

\* @brief Чисто виртуальный метод для решения второго задания

\*/

virtual Matrix task2() = 0;

};

#pragma once

#include "basicTask.h"

struct Tasks : public BasicTask

{

private:

Matrix matrix;

public:

/\*\*

\* @brief Инициализация объекта типа Tasks

\* @param matrix Матрица с которой надо выполнить задания

\*/

Tasks(Matrix matrix);

/\*\*

\* @brief function which replace the minimal modulus element with zero in every column

\* @return Мchanged matrix

\*/

Matrix task1() override;

/\*\*

\* @brief function which delete columns where first element more than last

\* @return changed matrix

\*/

Matrix task2() override;

};

#include "task.h"

#include "algorithm"

Tasks::Tasks(Matrix matrix)

:matrix(matrix)

{

}

Matrix Tasks::task1()

{

Matrix result{this->matrix};

result.transpose();

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

{

std::replace (matrix[i].row.begin(), matrix[i].row.end(), matrix[i].searchMinimun(), 0);

}

result.transpose();

return result;

}

Matrix Tasks::task2()

{

Matrix result {this->matrix};

result.transpose();

for(auto it = result.getMatrix().cbegin(); it !=result.getMatrix().cend(); it++)

{

if (it->row[0] > it->row[matrix.getSize()])

{

result.getMatrix().erase(it);

}

}

result.transpose();

return result;

}

**Tecты:**

#include <cstddef>

#include <gtest/gtest.h>

#include "matrix.h"

#include "matrix.cpp"

TEST(MatrixTest, TransposeTest) {

Matrix matrix{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, 3, 3};

matrix.transpose();

// Check the transposed matrix

EXPECT\_EQ(matrix[0][0], 1);

EXPECT\_EQ(matrix[0][1], 4);

EXPECT\_EQ(matrix[0][2], 7);

EXPECT\_EQ(matrix[1][0], 2);

EXPECT\_EQ(matrix[1][1], 5);

EXPECT\_EQ(matrix[1][2], 8);

EXPECT\_EQ(matrix[2][0], 3);

EXPECT\_EQ(matrix[2][1], 6);

EXPECT\_EQ(matrix[2][2], 9);

}

TEST(MatrixTest, DeleteColumnTest) {

Matrix matrix{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, 3, 3};

matrix.deleteColumn(1);

// Check the matrix after deleting a column

EXPECT\_EQ(matrix[0][0], 1);

EXPECT\_EQ(matrix[0][1], 3);

EXPECT\_EQ(matrix[1][0], 4);

EXPECT\_EQ(matrix[1][1], 6);

EXPECT\_EQ(matrix[2][0], 7);

EXPECT\_EQ(matrix[2][1], 9);

}

TEST(MatrixTest, DeleteRowTest) {

Matrix matrix{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, 3, 3};

matrix.deleteRow(1);

// Check the matrix after deleting a row

EXPECT\_EQ(matrix[0][0], 1);

EXPECT\_EQ(matrix[0][1], 2);

EXPECT\_EQ(matrix[1][0], 7);

EXPECT\_EQ(matrix[1][1], 8);

EXPECT\_EQ(matrix[2][0], 3);

EXPECT\_EQ(matrix[2][1], 9);

}

TEST(TasksTest, Task1Test) {

Matrix inputMatrix{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, 3, 3};

Tasks tasks(inputMatrix);

Matrix result = tasks.task1();

// Check the result after Task 1

EXPECT\_EQ(result[0][0], 0);

EXPECT\_EQ(result[0][1], 2);

EXPECT\_EQ(result[0][2], 0);

EXPECT\_EQ(result[1][0], 0);

EXPECT\_EQ(result[1][1], 5);

EXPECT\_EQ(result[1][2], 0);

EXPECT\_EQ(result[2][0], 0);

EXPECT\_EQ(result[2][1], 8);

EXPECT\_EQ(result[2][2], 0);

}

TEST(TasksTest, Task2Test) {

Matrix inputMatrix{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, 3, 3};

Tasks tasks(inputMatrix);

Matrix result = tasks.task2();

// Check the result after Task 2

EXPECT\_EQ(result.getSize(), 2); // Assuming two rows meet the condition

EXPECT\_EQ(result[0][0], 1);

EXPECT\_EQ(result[0][1], 2);

EXPECT\_EQ(result[0][2], 3);

EXPECT\_EQ(result[1][0], 7);

EXPECT\_EQ(result[1][1], 8);

EXPECT\_EQ(result[1][2], 9);

}

int main(int argc, char \*\*argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

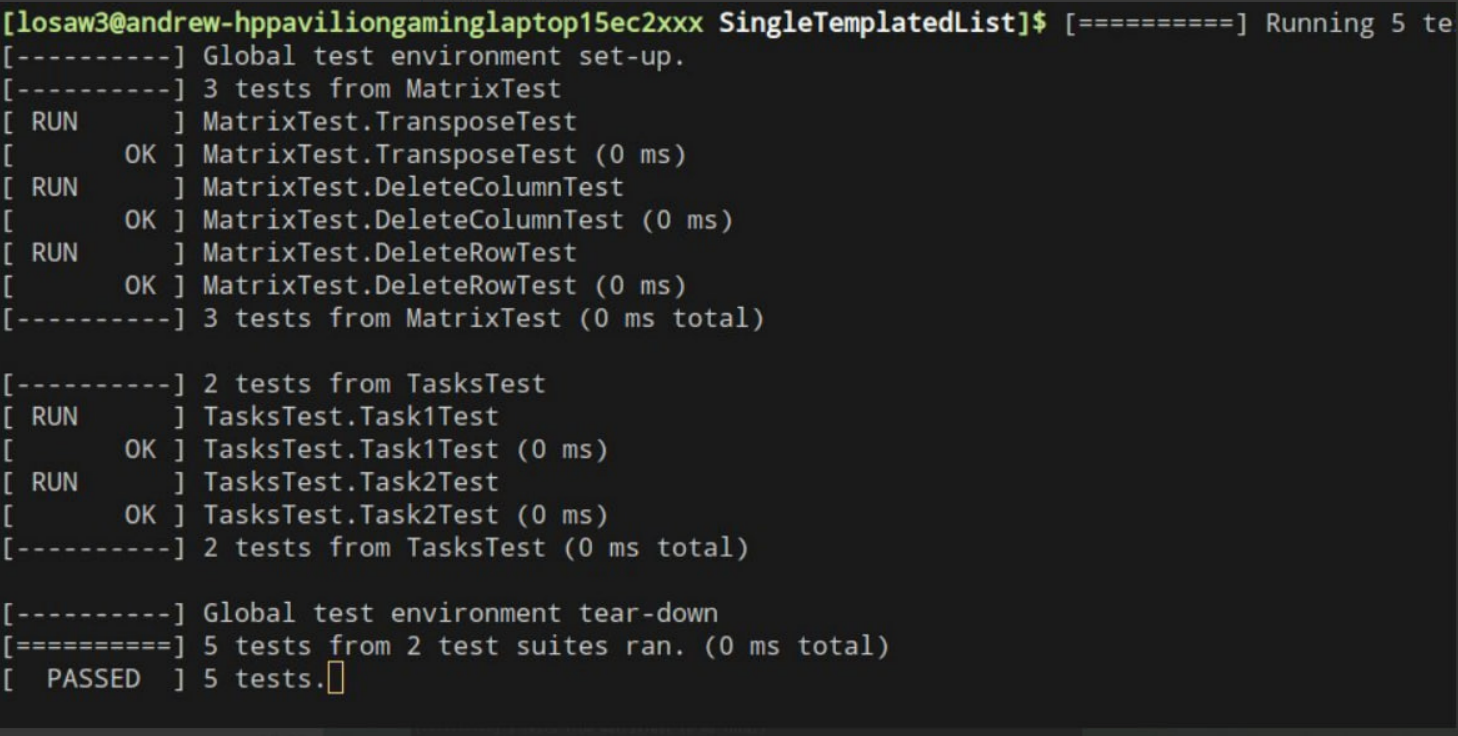


Рисунок 1 – прохождение тестов

**Uml – диаграмма:**

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

Автоматически созданное описание**

Рисунок 2 – диаграмма проекта

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – approve задания