**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ**

**Институт компьютерных технологий и информационной безопасности**

**Кафедра математического обеспечения и применения**

Отчет по лабораторной работе №2

по курсу «Объектно-ориентированное программирование»

**«Наследование в С++»**

Выполнил:

Студент гр.КТбо2-7

Серченко О. С.

Проверил:

Преподаватель по ОСиСПО

Тарасов С. А.

Таганрог 2020

Задание

Определить класс вещественных матриц (Matrix) с методами, реализующими сложение и умножение матриц, транспонирование. Также класс квадратичных матриц SquareMatrux с возможностью нахождения детерминанта.

Спецификация классов

class Matrix

{

public:

int GetRows() const;

int GetCols() const;

//void Randomize() const;

virtual Matrix& operator ~();

Matrix operator +(const Matrix& m) const;

Matrix operator -(const Matrix& m) const;

Matrix operator =(const Matrix& m) const;

Matrix operator \*(const Matrix& m) const;

double\* operator [](unsigned int row) const;

Matrix(unsigned int rows, unsigned int cols);

Matrix(const Matrix& m);

~Matrix();

protected:

unsigned int \_rows = 0;

unsigned int \_cols = 0;

double\*\* \_Matrix = nullptr;

};

class SquareMatrix :public Matrix

{

public:

Matrix& operator ~() override;

double Det() const;

SquareMatrix(unsigned int rowscols);

SquareMatrix(const Matrix& m);

~SquareMatrix() = default;

private:

void getCofactor(const SquareMatrix\* sm, SquareMatrix\* tmp, int excessRow, int excessCol) const;

};

class FactoryMatrix

{

public:

static Matrix\* CreateMatrix(unsigned int rows, unsigned int cols);

};

class MatrixInteractor

{

public:

int RunFileMenu();

int RunConsoleMenu();

MatrixInteractor() = default;

MatrixInteractor(const MatrixInteractor&) = default;

~MatrixInteractor();

private:

void Expand(std::istream& is, bool prompt);

void MatrixSum();

void MatrixSub();

void MatrixMul();

void MatrixTran();

void MatrixDet();

void MatrixPrint();

void MatrixFindNo();

void MatrixFindDet();

void MatrixErase();

std::string \_fileName;

std::fstream \_file;

std::vector<Matrix\*> \_matrixVector;

std::map<std::pair<unsigned int, unsigned int>, std::set<unsigned int>> \_mapNo;

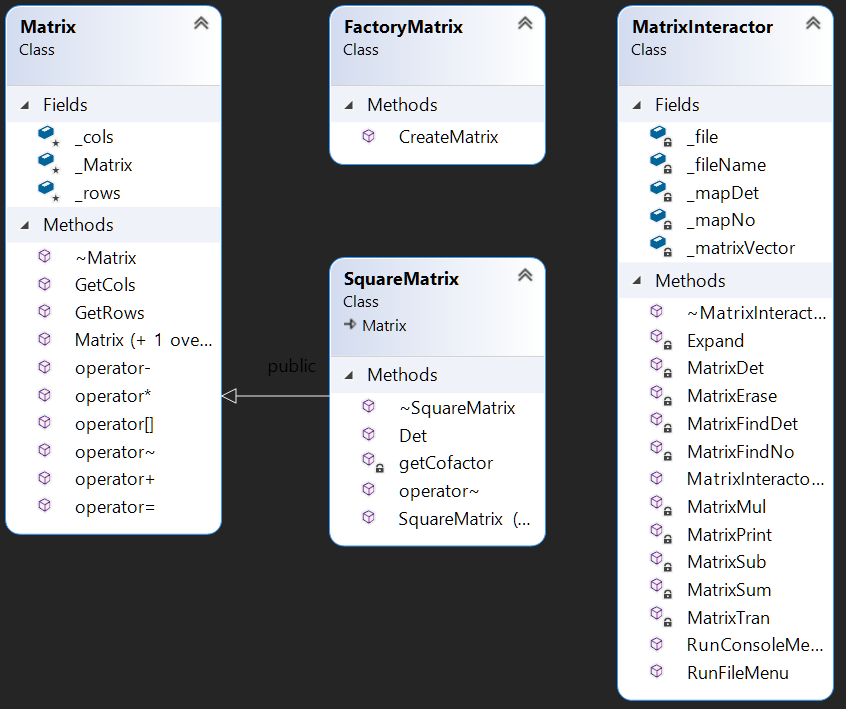
std::map<double, std::set<unsigned int>> \_mapDet;

};

Зависимости и алгоритмы

Использованы стандартные алгоритмы.

Диаграмма классов



Листинг

Lab1\_Serchenko\_Matrix.cpp:

#include "MatrixInteractor.h"

#include <vld.h>

#include <iostream>

int main()

{

try {

Run();

}

catch (std::exception e) {

std::cout << e.what() << std::endl;

getchar(); getchar();

}

return 0;

}

Matrix.h:

#pragma once

class Matrix

{

public:

int GetRows() const;

int GetCols() const;

//void Randomize() const;

virtual Matrix& operator ~();

Matrix operator +(const Matrix& m) const;

Matrix operator -(const Matrix& m) const;

Matrix operator =(const Matrix& m) const;

Matrix operator \*(const Matrix& m) const;

double\* operator [](unsigned int row) const;

Matrix(unsigned int rows, unsigned int cols);

Matrix(const Matrix& m);

~Matrix();

protected:

unsigned int \_rows = 0;

unsigned int \_cols = 0;

double\*\* \_Matrix = nullptr;

};

class SquareMatrix :public Matrix

{

public:

Matrix& operator ~() override;

double Det() const;

SquareMatrix(unsigned int rowscols);

SquareMatrix(const Matrix& m);

~SquareMatrix() = default;

private:

void getCofactor(const SquareMatrix\* sm, SquareMatrix\* tmp, int excessRow, int excessCol) const;

};

class FactoryMatrix

{

public:

static Matrix\* CreateMatrix(unsigned int rows, unsigned int cols);

};

Matrix.cpp:

#include "Matrix.h"

#include <algorithm>

#include <stdexcept>

int Matrix::GetRows() const {

return \_rows;

}

int Matrix::GetCols() const {

return \_cols;

}

/\*void Matrix::Randomize() const

{

for (unsigned int i = 0; i < \_rows; i++)

{

for (unsigned int j = 0; j < \_cols; j++)

{

\_Matrix[i][j] = std::rand() % 10;

}

}

}\*/

Matrix& Matrix::operator~()

{

double\*\* \_NewMatrix = new double\* [\_cols];

for (unsigned int i = 0; i < \_cols; i++)

{

\_NewMatrix[i] = new double[\_rows];

for (unsigned int j = 0; j < \_rows; j++)

{

\_NewMatrix[i][j] = \_Matrix[j][i];

}

}

this->~Matrix();

std::swap(\_rows, \_cols);

\_Matrix = \_NewMatrix;

return \*this;

}

Matrix Matrix::operator+(const Matrix& m) const

{

if (\_rows != m.\_rows || \_cols != m.\_cols)

{

throw std::length\_error("Sizes of matrices differ");

}

Matrix tmp(\_rows, \_cols);

for (unsigned int i = 0; i < \_rows; i++)

{

for (unsigned int j = 0; j < \_cols; j++)

{

(tmp).\_Matrix[i][j] = \_Matrix[i][j] + m.\_Matrix[i][j];

}

}

return tmp;

}

Matrix Matrix::operator-(const Matrix& m) const

{

if (\_rows != m.\_rows || \_cols != m.\_cols)

{

throw std::length\_error("Sizes of matrices differ");

}

Matrix tmp(\_rows, \_cols);

for (unsigned int i = 0; i < \_rows; i++)

{

for (unsigned int j = 0; j < \_cols; j++)

{

tmp.\_Matrix[i][j] = \_Matrix[i][j] - m.\_Matrix[i][j];

}

}

return tmp;

}

Matrix Matrix::operator=(const Matrix& m) const

{

if (\_rows != m.\_rows || \_cols != m.\_cols)

{

throw std::length\_error("Sizes of matrices differ");

}

for (unsigned int i = 0; i < \_rows; i++)

{

std::copy(m.\_Matrix[i], m.\_Matrix[i] + \_cols, \_Matrix[i]);

}

return \*this;

}

Matrix Matrix::operator\*(const Matrix& m) const

{

if (\_cols != m.\_rows)

{

throw std::length\_error("1st multiplier's cols != 2nd multiplier's rows");

}

Matrix tmp(\_rows, m.\_cols);

unsigned int i, j, k;

for (i = 0; i < \_rows; i++)

{

for (j = 0; j < m.\_cols; j++)

{

tmp.\_Matrix[i][j] = 0;

for (k = 0; k < \_cols; k++)

{

tmp.\_Matrix[i][j] += \_Matrix[i][k] \* m.\_Matrix[k][j];

}

}

}

return tmp;

}

double\* Matrix::operator[](unsigned int row)const

{

return \_Matrix[row];

}

Matrix::Matrix(unsigned int rows, unsigned int cols)

{

\_rows = rows;

\_cols = cols;

\_Matrix = new double\* [\_rows];

for (unsigned int i = 0; i < \_rows; i++)

{

\_Matrix[i] = new double[\_cols];

}

}

Matrix::Matrix(const Matrix& m)

{

\_rows = m.\_rows;

\_cols = m.\_cols;

\_Matrix = new double\*[\_rows];

for (unsigned int i = 0; i < \_rows; i++)

{

\_Matrix[i] = new double[\_cols];

std::copy(m.\_Matrix[i], m.\_Matrix[i] + \_cols , \_Matrix[i]);

}

}

Matrix::~Matrix()

{

for (unsigned int i = 0; i < \_rows; i++)

{

delete[] \_Matrix[i];

}

delete[] \_Matrix;

}

SquareMatrix::SquareMatrix(unsigned int rowscols)

:Matrix(rowscols, rowscols) {}

SquareMatrix::SquareMatrix(const Matrix& m)

:Matrix(m)

{

if (m.GetCols() != m.GetRows())

{

throw std::invalid\_argument("SquareMatrix creation failed");

}

}

Matrix& SquareMatrix::operator~()

{

for (unsigned int i = 0; i < \_rows; i++)

{

for (unsigned int j = i + 1; j < \_cols; j++)

{

std::swap(\_Matrix[i][j], \_Matrix[j][i]);

}

}

return \*this;

}

double SquareMatrix::Det() const

{

double D = 0, n = this->GetRows();

if (n == 0)

{

return 0;

}

if (n == 1)

{

return (\*this)[0][0];

}

SquareMatrix\* tmp = new SquareMatrix(n - 1);

int sign = 1;

for (int f = 0; f < n; f++)

{

getCofactor(this, tmp, 0, f);

D += sign \* (\*this)[0][f] \* (\*tmp).Det();

sign = -sign;

}

delete tmp;

return D;

}

void SquareMatrix::getCofactor(const SquareMatrix\* sm, SquareMatrix\* tmp, int excessRow, int excessCol) const

{

int i = 0, j = 0, n = (\*sm).GetRows();

for (int row = 0; row < n; row++)

{

for (int col = 0; col < n; col++)

{

if (row != excessRow && col != excessCol)

{

(\*tmp)[i][j++] = (\*sm)[row][col];

if (j == n - 1)

{

j = 0;

i++;

}

}

}

}

}

Matrix\* FactoryMatrix::CreateMatrix(unsigned int rows, unsigned int cols)

{

if (rows == cols)

{

return new SquareMatrix(rows);

}

return new Matrix(rows, cols);

}

MatrixInteractor.h:

#pragma once

#include "Matrix.h"

#include <fstream>

#include <vector>

#include <map>

#include <set>

class MatrixInteractor

{

public:

int RunFileMenu();

int RunConsoleMenu();

MatrixInteractor() = default;

MatrixInteractor(const MatrixInteractor&) = default;

~MatrixInteractor();

private:

void Expand(std::istream& is, bool prompt);

void MatrixSum();

void MatrixSub();

void MatrixMul();

void MatrixTran();

void MatrixDet();

void MatrixPrint();

void MatrixFindNo();

void MatrixFindDet();

void MatrixErase();

std::string \_fileName;

std::fstream \_file;

std::vector<Matrix\*> \_matrixVector;

std::map<std::pair<unsigned int, unsigned int>, std::set<unsigned int>> \_mapNo;

std::map<double, std::set<unsigned int>> \_mapDet;

};

void Run();

int RunMainMenu();

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

MatrixInteractor.cpp:

#include "MatrixInteractor.h"

#include <iostream>

#define Check(No) No >= \_matrixVector.size()

#define ScanCheck3(first, second, result) \

std::cout << "Введите номер первой матрицы" << std::endl;\

std::cin >> first;\

std::cout << "Введите номер второй матрицы" << std::endl;\

std::cin >> second;\

std::cout << "Введите номер результирующей матрицы" << std::endl;\

std::cin >> result;\

if(Check(first) || Check(second) || Check(result))throw std::out\_of\_range("Wrong Parameters");

#define ScanCheck1(No) std::cout << "Введите номер матрицы" << std::endl;\

std::cin >> No;\

if(Check(No)) throw std::out\_of\_range("Wrong Parameters");

void Run()

{

setlocale(LC\_ALL, "Russian");

while (RunMainMenu() != 0);

}

int RunMainMenu()

{

system("cls");

std::cout << "Функционал:" << std::endl;

std::cout << "1 - Файловый ввод матрицы" << std::endl;

std::cout << "2 - Консольный ввод матрицы" << std::endl;

std::cout << "0 - Выход" << std::endl;

MatrixInteractor MI;

int choice;

std::cin >> choice;

switch (choice)

{

case 1:

while (MI.RunFileMenu() != 0);

break;

case 2:

while (MI.RunConsoleMenu() != 0);

break;

default:

return 0;

}

return 1;

}

int MatrixInteractor::RunFileMenu()

{

std::cout << "1 - Ввести имя файла и прочитать содержимое" << std::endl;

std::cout << "2 - Произвести сложение" << std::endl;

std::cout << "3 - Произвести вычитание" << std::endl;

std::cout << "4 - Произвести умножение" << std::endl;

std::cout << "5 - Произести транспонирование" << std::endl;

std::cout << "6 - Вывести det" << std::endl;

std::cout << "7 - Создать новую матрицу" << std::endl;

std::cout << "8 - Вывести матрицу на экран" << std::endl;

std::cout << "9 - Показать номера матриц по размерности" << std::endl;

std::cout << "10 - Показать номера матриц по det" << std::endl;

std::cout << "11 - Удалить матрицу" << std::endl;

std::cout << "0 - Выйти и вывести данные" << std::endl;

int choice;

std::cin >> choice;

if (!\_file.is\_open() && choice != 1)

{

throw std::logic\_error("Имя файла не задано");

}

switch (choice)

{

case 1:

{

if (\_file.is\_open())

{

this->~MatrixInteractor();

}

std::cout << "Имя файла:" << std::endl;

std::cin >> \_fileName;

\_file.open(\_fileName, std::ios\_base::in);

if (!\_file.is\_open())

{

throw std::invalid\_argument("Неверное имя файла");

}

unsigned int quantity;

\_file >> quantity;

for (unsigned int count = 0; count < quantity; count++)

{

Expand(\_file, false);

}

break;

}

case 2:

{

MatrixSum();

break;

}

case 3:

{

MatrixSub();

break;

}

case 4:

{

MatrixMul();

break;

}

case 5:

{

MatrixTran();

break;

}

case 6:

{

MatrixDet();

break;

}

case 7:

{

Expand(std::cin, true);

break;

}

case 8:

{

MatrixPrint();

break;

}

case 9:

{

MatrixFindNo();

break;

}

case 10:

{

MatrixFindDet();

break;

}

case 11:

{

MatrixErase();

break;

}

case 0:

{

std::cout << "Введите имя файла приёмника" << std::endl;

std::cin >> \_fileName;

\_file.close();

\_file.open(\_fileName, std::ios\_base::trunc | std::ios\_base::out);

if (!\_file.is\_open())

{

throw std::invalid\_argument("Неверное имя файла");

}

\_file << \_matrixVector.size() << std::endl;

for (auto elem : \_matrixVector)

{

\_file << \*elem;

}

return 0;

}

}

return 1;

}

int MatrixInteractor::RunConsoleMenu()

{

std::cout << "1 - Создать новую матрицу" << std::endl;

std::cout << "2 - Произвести сложение " << std::endl;

std::cout << "3 - Произвести вычитание" << std::endl;

std::cout << "4 - Произвести умножение" << std::endl;

std::cout << "5 - Произести транспонирование" << std::endl;

std::cout << "6 - Вывести det" << std::endl;

std::cout << "7 - Вывести матрицу на экран" << std::endl;

std::cout << "8 - Показать номера матриц по размерности" << std::endl;

std::cout << "9 - Показать номера матриц по det" << std::endl;

std::cout << "10 - Удалить матрицу" << std::endl;

std::cout << "0 - Выйти" << std::endl;

int choice;

std::cin >> choice;

switch (choice)

{

case 1:

{

Expand(std::cin, true);

break;

}

case 2:

{

MatrixSum();

break;

}

case 3:

{

MatrixSub();

break;

}

case 4:

{

MatrixMul();

break;

}

case 5:

{

MatrixTran();

break;

}

case 6:

{

MatrixDet();

break;

}

case 7:

{

MatrixPrint();

break;

}

case 8:

{

MatrixFindNo();

break;

}

case 9:

{

MatrixFindDet();

break;

}

case 10:

{

MatrixErase();

break;

}

case 0:

{

return 0;

}

}

return 1;

}

void MatrixInteractor::Expand(std::istream& is, bool prompt)

{

unsigned int rows, cols;

if (prompt)std::cout << "Введите количество рядов" << std::endl;

is >> rows;

if (prompt)std::cout << "Введите количество колонн" << std::endl;

is >> cols;

\_mapNo[std::pair<unsigned int, unsigned int>(rows, cols)].insert(\_matrixVector.size());

\_matrixVector.push\_back(FactoryMatrix::CreateMatrix(rows, cols));

if (prompt)std::cout << "Введите матрицу " << rows << " на " << cols << std::endl;

for (unsigned int i = 0; i < rows; i++)

{

double\* row = (\*\_matrixVector.back())[i]; //Указатель на очередной ряд

for (unsigned int j = 0; j < cols; j++)

{

is >> row[j];

}

}

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector.back()))).Det()].insert(\_matrixVector.size() - 1);

}

}

void MatrixInteractor::MatrixSum()

{

unsigned int first, second, result, rows, cols;

ScanCheck3(first, second, result)

rows = \_matrixVector[result]->GetRows();

cols = \_matrixVector[result]->GetCols();

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].erase(result);

}

\*(\_matrixVector[result]) = \*(\_matrixVector[first]) + \*(\_matrixVector[second]);

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].insert(result);

}

}

void MatrixInteractor::MatrixSub()

{

unsigned int first, second, result, rows, cols;

ScanCheck3(first, second, result)

rows = \_matrixVector[result]->GetRows();

cols = \_matrixVector[result]->GetCols();

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].erase(result);

}

\*(\_matrixVector[result]) = \*(\_matrixVector[first]) - \*(\_matrixVector[second]);

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].insert(result);

}

}

void MatrixInteractor::MatrixMul()

{

unsigned int first, second, result, rows, cols;

ScanCheck3(first, second, result)

rows = \_matrixVector[result]->GetRows();

cols = \_matrixVector[result]->GetCols();

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].erase(result);

}

\*(\_matrixVector[result]) = \*(\_matrixVector[first]) \* \*(\_matrixVector[second]);

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*(\_matrixVector[result]))).Det()].insert(result);

}

}

void MatrixInteractor::MatrixTran()

{

unsigned int No;

ScanCheck1(No)

~(\*(\_matrixVector[No]));

}

void MatrixInteractor::MatrixDet()

{

unsigned int No;

ScanCheck1(No)

std::cout << ((SquareMatrix)(\*(\_matrixVector[No]))).Det() << std::endl;

}

void MatrixInteractor::MatrixPrint()

{

unsigned int No;

ScanCheck1(No)

std::cout << \*(\_matrixVector[No]);

}

void MatrixInteractor::MatrixFindNo()

{

unsigned int rows, cols;

std::cout << "Введите количество рядов" << std::endl;

std::cin >> rows;

std::cout << "Введите количество колонн" << std::endl;

std::cin >> cols;

for (auto elem : \_mapNo[std::pair<unsigned int, unsigned int>(rows, cols)])

{

std::cout << elem << ' ';

}

std::cout << std::endl;

}

void MatrixInteractor::MatrixFindDet()

{

double det;

std::cout << "Введите значение det" << std::endl;

std::cin >> det;

for (auto elem : \_mapDet[det])

{

std::cout << elem << ' ';

}

std::cout << std::endl;

}

void MatrixInteractor::MatrixErase()

{

unsigned int No;

ScanCheck1(No)

std::vector<Matrix\*>::iterator vecIt = \_matrixVector.begin() + No;

unsigned int rows = (\*vecIt)->GetRows(), cols = (\*vecIt)->GetCols();

\_mapNo[std::pair<unsigned int, unsigned int>(rows, cols)].erase(No);

if (rows == cols)

{

\_mapDet[((SquareMatrix)(\*\*vecIt)).Det()].erase(No);

}

delete (\*vecIt);

\_matrixVector.erase(vecIt);

}

MatrixInteractor::~MatrixInteractor()

{

for (auto elem : \_matrixVector)

{

delete elem;

}

}

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

{

int i, j;

int rows = m.GetRows(), cols = m.GetCols();

os << rows << ' ' << cols << std::endl;

if (rows == 0 || cols == 0)

{

return os;

}

for (i = 0; i < rows; i++)

{

double\* row = m[i];

for (j = 0; j < cols - 1; j++)

{

os << row[j] << ' ';

}

os << row[j] << std::endl;

}

return os;

}