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 ЭВМ

**Лабораторная работа № 1**

по курсу «Объектно-ориентированное программирование»

**«Классы и объекты в C++»**

**8 ВАРИАНТ**

Выполнили:

студенты гр. КТбо1-7

Миронченко П.Д.

Проверил:

Тарасов С. А.

**Таганрог 2020**

1. **Вариант задания №8**

Определить класс вещественных матриц (Matrix) с методами, реализующими сложение и умножение матриц, транспонирование. Размер матрицы задается при ее создании.

1. **Спецификация класса Matrix**

Класс Matrix содержит 3 приватных поля \_colsLen, \_rowsLen и \_A, а также методы для взаимодействия с ними, такие как: получение количества строк и столбцов, получение изменение поля матрицы по индексам, сложение и умножение матриц и транспонирование матрицы. В добавок, были перегружены операторы ввода и вывода для упрощения взаимодействия в консоли.

1. **Используемые математические зависимости и алгоритмы**

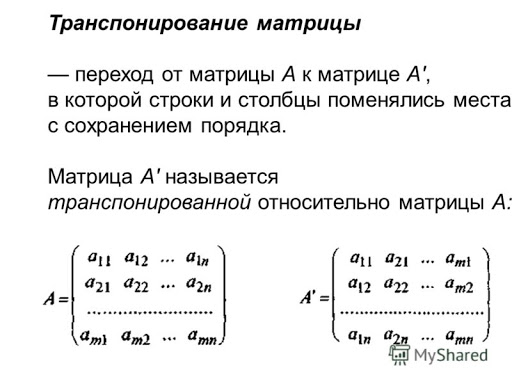
В этой лабораторной работе использовались алгоритмы сложения, умножения матриц и транспонирования матрицы.

A screenshot of a cell phone

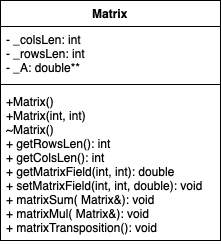
Description automatically generated

A screenshot of a cell phone

Description automatically generated



1. **Диаграмма класса**



1. **Листинг программы**

**lab1.cpp**

#include <iostream>

#include "interactor.h"

using namespace std;

int main()

{

Interactor it;

it.readObjects("input.txt");

freopen("commands.txt", "r", stdin);

int n;

cin >> n;

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

{

char ch;

int tar;

cin >> ch >> tar;

cout << it.interaction(ch, tar, i+1) << endl;

}

cout << it << endl;

return 0;

}

**interactor.h**

#ifndef INTERACTOR\_H

#define INTERACTOR\_H

#include "matrix.h"

class Interactor

{

private:

Matrix\* \_first;

Matrix\* \_second;

public:

Interactor();

void readObjects(std::string);

std::string interaction(char, int, int);

friend std::ostream& operator<<(std::ostream& , const Interactor&);

~Interactor();

};

#endif

**matrix.h**

#ifndef MATRIX\_H

#define MATRIX\_H

#include <istream>

#include <ostream>

class Matrix

{

int \_colsLen, \_rowsLen;

double\*\* \_A;

double\*\* \_buildMatrix(int, int);

void \_destroyMatrix(double\*\* , int);

public:

Matrix();

Matrix(int, int);

int getColsLen() const;

int getRowsLen() const;

double& matrixField(int i, int j);

void matrixSum(const Matrix&);

void matrixMul(const Matrix&);

void matrixTransposition();

friend std::istream& operator>>(std::istream&, const Matrix&);

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

~Matrix();

};

#endif

**matrix.cpp**

#include <istream>

#include <ostream>

#include "matrix.h"

double\*\* Matrix::\_buildMatrix(int n, int m)

{

double\*\* array = new double\*[n];

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

{

array[i] = new double[m];

for (int j = 0; j < m; ++j)

{

array[i][j] = 0;

}

}

return array;

}

void Matrix::\_destroyMatrix(double\*\* matrix, int n)

{

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

{

delete[] matrix[i];

}

if (matrix != nullptr)

{

delete[] matrix;

}

}

Matrix::Matrix()

{

\_colsLen = 0;

\_rowsLen = 0;

\_A = nullptr;

}

Matrix::Matrix(int n, int m)

{

\_colsLen = n;

\_rowsLen = m;

\_A = \_buildMatrix(n,m);

}

int Matrix::getColsLen() const

{

return \_colsLen;

}

int Matrix::getRowsLen() const

{

return \_rowsLen;

}

double& Matrix::matrixField(int i, int j)

{

return \_A[i][j];

}

void Matrix::matrixSum(const Matrix& term)

{

if (\_colsLen != term.getColsLen() || \_rowsLen != term.getRowsLen())

{

throw std::logic\_error("Another matrix has a different shape\n");

}

if (\_rowsLen == 0 || \_rowsLen == 0 || term.getColsLen() == 0 || term.getRowsLen() == 0)

{

throw std::length\_error("Empty matrix\n");

}

for (int i = 0; i < \_colsLen; ++i)

{

for (int j = 0; j < \_rowsLen; ++j)

{

\_A[i][j] += term.\_A[i][j];

}

}

}

void Matrix::matrixMul(const Matrix& factor)

{

if (\_rowsLen != factor.getColsLen())

{

throw std::logic\_error("Rows of one matrix are not equal to columns of another\n");

}

if (\_rowsLen == 0 || \_rowsLen == 0 || factor.getColsLen() == 0 || factor.getRowsLen() == 0)

{

throw std::length\_error("Empty matrix\n");

}

double\*\* A\_new = \_buildMatrix(\_colsLen, factor.getRowsLen());

for (int i = 0; i < \_colsLen; ++i)

{

for (int j = 0; j < factor.getRowsLen(); ++j)

{

A\_new[i][j] = 0;

for (int k = 0; k < \_rowsLen; ++k)

{

A\_new[i][j] += \_A[i][k]\*factor.\_A[k][j];

}

}

}

\_destroyMatrix(\_A, \_colsLen);

\_A = A\_new;

\_rowsLen = factor.getRowsLen();

}

void Matrix::matrixTransposition()

{

if (\_rowsLen == 0 || \_rowsLen == 0)

{

throw std::length\_error("Empty matrix\n");

}

double\*\* A\_new = \_buildMatrix(\_rowsLen,\_colsLen);

for (int i = 0; i < \_rowsLen; ++i)

{

for (int j = 0; j < \_colsLen; ++j)

{

A\_new[i][j] = \_A[j][i];

}

}

\_destroyMatrix(\_A, \_colsLen);

\_A = A\_new;

std::swap(\_rowsLen,\_colsLen);

}

std::istream& operator>>(std::istream& in, const Matrix& object)

{

for (int i = 0; i < object.\_colsLen;++i)

{

for (int j = 0; j < object.\_rowsLen; ++j)

{

in >> object.\_A[i][j];

}

}

return in;

}

std::ostream& operator<<(std::ostream& out, const Matrix& object)

{

out << "-----------------------------\n";

out << "Matrix has "

<< object.\_colsLen

<< " rows and "

<< object.\_rowsLen

<< " cols\n";

for (int i = 0; i < object.\_colsLen;++i)

{

for (int j = 0; j < object.\_rowsLen; ++j)

{

out << object.\_A[i][j] << ' ';

}

out << '\n';

}

out << "-----------------------------\n";

return out;

}

Matrix::~Matrix()

{

\_destroyMatrix(\_A, \_colsLen);

}

**interactor.cpp**

#include <fstream>

#include <ostream>

#include <string>

#include <sstream>

#include "interactor.h"

Interactor::Interactor(): \_first(nullptr), \_second(nullptr) {}

void Interactor::readObjects(std::string file)

{

std::ifstream fin(file);

int x,y;

fin >> x >> y;

if (\_first != nullptr) delete \_first;

\_first = new Matrix(x,y);

fin >> \*\_first;

fin >> x >> y;

if (\_second != nullptr) delete \_second;

\_second = new Matrix(x,y);

fin >> \*\_second;

fin.close();

}

std::string Interactor::interaction(char command, int target, int cnt)

{

if (command != '+' && command != '\*' && command != 'T' )

{

return "Bad command";

}

if (target < 1 || target > 2)

{

return "Bad target";

}

if (target == 2)

{

std::swap(\_first,\_second);

}

std::stringstream result;

result << cnt << ")\n";

try

{

if (command == '+')

{

\_first->matrixSum(\*\_second);

}

else if (command == '\*')

{

\_first->matrixMul(\*\_second);

}

else if (command == 'T')

{

\_first->matrixTransposition();

}

}

catch (std::logic\_error e)

{

result << e.what() << '\n';

return result.str();

}

catch (std::length\_error e)

{

result << e.what() << '\n';

return result.str();

}

result << \*\_first;

if (target == 2)

{

std::swap(\_first,\_second);

}

return result.str();

}

std::ostream& operator<<(std::ostream& out, const Interactor& object)

{

out << "///////////////////////////////////\n";

if (object.\_first == nullptr)

{

out << "First matrix has no elements\n";

}

else

{

out << \*object.\_first;

}

if (object.\_second == nullptr)

{

out << "Second matrix has no elements\n";

}

else

{

out << \*object.\_second;

}

out << "///////////////////////////////////\n";

return out;

}

Interactor::~Interactor()

{

if (\_first != nullptr) delete \_first;

if (\_second!= nullptr) delete \_second;

}