ФЕДЕРАЛЬНОЕ АГЕНСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ПУТЕЙ СООБЩЕНИЯ Императора Александра I»

Кафедра «Информационные и вычислительные системы»

Дисциплина «Программирование С++»

**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

ВАРИАНТ 19

|  |  |
| --- | --- |
| Выполнил студент  Факультет: АИТ  Группа: ИВБ-211 | Шефнер А. |
| Проверил: | Проузин О.В. |

**Санкт-Петербург**

**2023**

Оценочный лист результатов ЛР № 4

Ф.И.О. студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шефнер Альберт\_\_\_\_\_

Группа \_\_\_\_\_\_\_\_\_\_\_\_\_ИВБ-211\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№**  **п/п** | **Материалы необходимые для оценки знаний, умений**  **и навыков** | **Показатель**  **оценивания** | **Критерии**  **Оценивания** | **Шкала оценивания** | **Оценка** |
| 1 | Лабораторная работа№ | Соответствие методике выполнения | Соответствует | 7 |  |
| Не соответствует | 0 |
| Срок выполнения | Выполнена в срок | 2 |  |
| Выполнена с опозданием на 2 недели | 0 |
| оформление | Соответствует требованиям | 1  0 |  |
| Не соответствует |  |
|  | **ИТОГО количество баллов** |  |  | 10 |  |

Доцент кафедры

«Информационные и вычислительные

системы» Проурзин О.В. «\_\_» \_\_\_\_\_\_\_\_\_\_2023 г.

**Цели работы:**

* Создание класса.
* Перегрузка операторов.
* Использование конструктора по умолчанию
* Использование параметризованного конструктора
* Использование конструктора копирование
* Определение свойств класса

**Задание**

1. Разработать авторский класс, который задаёт формат заданных объектов. Вариант *индивидуального авторского класса* выбирается в файле «Варианты\_Классы.docx» исходя из *номера студента в списке группы.*

Требования к членам-данным класса:

* спецификатор доступа к членам-данным должен быть **private**;

Требования к методам класса:

* спецификатор доступа к методам должен быть **public**;
* методы изменения значения **(set…)** для каждого члена-данного;
* методы, возвращающих значение **(get…)**, для каждого члена-данного;
* конструкторы по умолчанию;
* конструктор с параметрами;
* конструктор копирования;
* деструктор класса;

Перегрузить операторы ввода-вывода, а также операторы

* + и – (варианты 1, 2, 5, 8);
* == (варианты 3, 4, 6, 7, 9, 10).

1. Создать тестовую функцию **main ( )**, в которой:
   1. Создать объект авторского класса явно (используя как конструктор по умолчанию, так и параметризованный конструктор), а также через указатель.
   2. При помощи оператора **switch** предусмотреть выбор блоков программы для проверки методов авторского класса:
      1. Создание и вывод состояния объекта авторского класса, используя явное объявление объекта и через указатель.
      2. Изменение значения члена-данного (предусмотреть отдельный блок для каждого члена-данного).
      3. Применение уникального метода класса (в зависимости от варианта индивидуального задания).
      4. Применение арифметических и операторов отношений к объектам класса (в зависимости от варианта индивидуального задания), а также операторов ввода-вывода.
      5. Выход из программы.

**Вариант 9**

Описать класс, реализующий тип данных «квадратная вещественная матрица 4х4» и работу с ними. Класс должен реализовывать следующие методы:

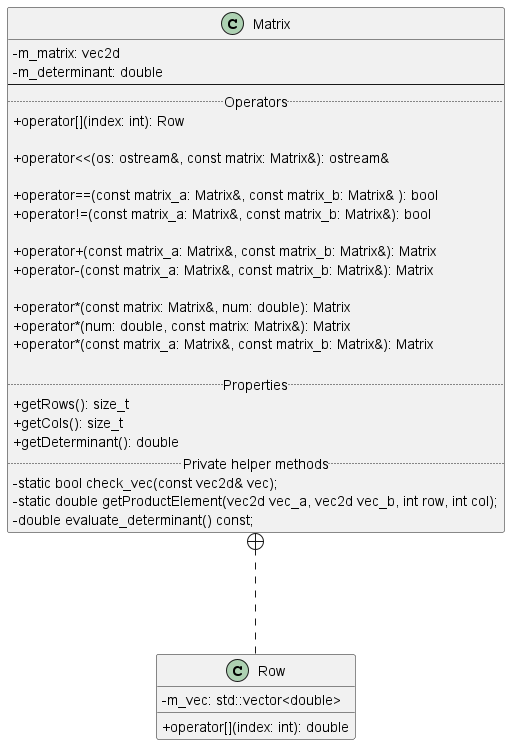
* умножение матрицы на число;
* транспонирование матрицы.

Написать программу, демонстрирующую работу с этим классом. Программа должна содержать меню, позволяющее осуществить проверку всех методов класса.

**Используемые средства**

В качестве интегрированной среды разработки использовалась JetBrains CLion.

Для работы в консоли с потоками ввода-вывода использовалась стандартная библиотека <iostream>

**UML-диаграмма программы:**

**Код программы**

**main.cpp**

#include <iostream>

#include <fstream>

#include "matrix.h"

int main() {

system("color F0");

std::ifstream fin(R"(D:\University\2 term\Programming C++\Laboratory work 4\cpp\_lab\_4\matrix.txt)");

int rows, cols;

fin >> rows >> cols;

vec2d vec;

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

std::vector<double> row;

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

int tmp;

fin >> tmp;

row.push\_back(tmp);

}

vec.push\_back(row);

}

Matrix m1(vec);

Matrix m2 = m1 + m1;

Matrix m3 = m1 \* m2;

std::cout << "Matrix m1:\n" << m1 << std::endl;

std::cout << "Determinant of m1: " << m1.getDeterminant() << std::endl << std::endl;

std::cout << "m2 = m1 + m1:\n" << std::endl;

std::cout << "m3 = m1 \* m2:\n" << m3 << std::endl;

std::cout << "Element of m3 at 3 row and 4 column: " << m3[2][3] << std::endl << std::endl;

system("pause");

return 0;

}

**matrix.h**

#ifndef CPP\_LAB\_4\_MATRIX\_H

#define CPP\_LAB\_4\_MATRIX\_H

#include <ostream>

#include <utility>

#include <vector>

typedef std::vector<std::vector<double>> vec2d;

class Matrix {

public:

explicit Matrix(const vec2d& vec);

Matrix(Matrix& matrix);

~Matrix();

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

friend bool operator ==(const Matrix& matrix\_a, const Matrix& matrix\_b);

friend bool operator !=(const Matrix& matrix\_a, const Matrix& matrix\_b) { return !(matrix\_a == matrix\_b); }

friend Matrix operator +(const Matrix& matrix\_a, const Matrix& matrix\_b);

friend Matrix operator -(const Matrix& matrix\_a, const Matrix& matrix\_b);

friend Matrix operator \*(const Matrix& matrix, double num);

friend Matrix operator \*(double num, const Matrix& matrix) { return matrix \* num; }

friend Matrix operator \*(const Matrix& matrix\_a, const Matrix& matrix\_b);

size\_t getRows() const { return m\_matrix.size(); }

size\_t getCols() const { return m\_matrix[0].size(); }

double getDeterminant();

class Row {

public:

explicit Row(std::vector<double>& vec) : m\_vec(vec) {}

double operator[](int index);

private:

std::vector<double> m\_vec;

};

Row operator[](int index);

private:

vec2d m\_matrix;

double m\_determinant;

static bool check\_vec(const vec2d& vec);

static double getProductElement(vec2d vec\_a, vec2d vec\_b, int row, int col);

double evaluate\_determinant() const;

};

#endif //CPP\_LAB\_4\_MATRIX\_H

**matrix.cpp**

#include "matrix.h"

#include <cmath>

#include "determinant.h"

#define DETERMINANT\_NOT\_CALCULATED "1"

Matrix::Matrix(const vec2d& vec) {

if(!check\_vec(vec)) {

exit(-1);

}

m\_matrix = (vec);

m\_determinant = std::nan(DETERMINANT\_NOT\_CALCULATED);

}

Matrix::Matrix(Matrix &matrix) {

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

std::vector<double> row;

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

row.push\_back(matrix.m\_matrix[i][j]);

}

m\_matrix.push\_back(row);

}

m\_determinant = matrix.m\_determinant;

}

Matrix::~Matrix() = default;

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

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

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

os << matrix.m\_matrix[i][j] << " ";

}

os << '\n';

}

return os;

}

bool Matrix::check\_vec(const vec2d& vec) {

if (vec.empty()) return false;

size\_t rowSize = vec[0].size();

for(const auto& row : vec) {

if(row.size() != rowSize) return false;

}

return true;

}

Matrix::Row Matrix::operator[](int index) {

if(index >= getRows() || index < 0) exit(-1);

return Row(m\_matrix[index]);

}

double Matrix::Row::operator[](int index) {

if(index >= m\_vec.size() || index < 0) exit(-1);

return m\_vec[index];

}

bool operator==(const Matrix &matrix\_a, const Matrix &matrix\_b) {

if(matrix\_a.getRows() != matrix\_b.getRows() || matrix\_a.getCols() != matrix\_b.getCols()) return false;

for(int i = 0; i < matrix\_a.getRows(); i++) {

for(int j = 0; j < matrix\_a.getCols(); j++) {

if(matrix\_a.m\_matrix[i][j] != matrix\_b.m\_matrix[i][j]) return false;

}

}

return true;

}

Matrix operator+(const Matrix &matrix\_a, const Matrix &matrix\_b) {

if(matrix\_a.getRows() != matrix\_b.getRows() || matrix\_a.getCols() != matrix\_b.getCols())

exit(-1);

vec2d newVec;

for(int i = 0; i < matrix\_a.getRows(); i++) {

std::vector<double> row;

for(int j = 0; j < matrix\_a.getCols(); j++) {

row.push\_back(matrix\_a.m\_matrix[i][j] + matrix\_b.m\_matrix[i][j]);

}

newVec.push\_back(row);

}

return Matrix(newVec);

}

Matrix operator-(const Matrix &matrix\_a, const Matrix &matrix\_b) {

if(matrix\_a.getRows() != matrix\_b.getRows() || matrix\_a.getCols() != matrix\_b.getCols())

exit(-1);

vec2d newVec;

for(int i = 0; i < matrix\_a.getRows(); i++) {

std::vector<double> row;

for(int j = 0; j < matrix\_a.getCols(); j++) {

row.push\_back(matrix\_a.m\_matrix[i][j] - matrix\_b.m\_matrix[i][j]);

}

newVec.push\_back(row);

}

return Matrix(newVec);

}

Matrix operator\*(const Matrix &matrix, double num) {

vec2d newVec;

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

std::vector<double> row;

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

row.push\_back(matrix.m\_matrix[i][j] \* num);

}

newVec.push\_back(row);

}

return Matrix(newVec);

}

Matrix operator\*(const Matrix &matrix\_a, const Matrix &matrix\_b) {

if(matrix\_a.getRows() != matrix\_b.getCols()) exit(-1);

vec2d newVec;

for(int i = 0; i < matrix\_a.getRows(); i++) {

std::vector<double> row;

for(int j = 0; j < matrix\_b.getRows(); j++) {

row.push\_back(Matrix::getProductElement(matrix\_a.m\_matrix, matrix\_b.m\_matrix, i, j));

}

newVec.push\_back(row);

}

return Matrix(newVec);

}

double Matrix::getProductElement(vec2d vec\_a, vec2d vec\_b, int row, int col) {

double element = 0;

for(int i = 0; i < vec\_b.size(); i++) {

element += vec\_a[row][i] \* vec\_b[i][col];

}

return element;

}

double Matrix::getDeterminant() {

if(std::isnan(m\_determinant)) {

if(getRows() != getCols()) exit(-1);

m\_determinant = evaluate\_determinant();

};

return m\_determinant;

}

double Matrix::evaluate\_determinant() const {

double\*\* matrix\_array = new double\*[getRows()];

for(int i = 0; i < getRows(); i++) {

matrix\_array[i] = new double[getRows()];

for(int j = 0; j < getRows(); j++) {

matrix\_array[i][j] = m\_matrix[i][j];

}

}

double det = eval\_determinant(matrix\_array, getRows());

for(int i = 0; i < getRows(); i++) {

delete matrix\_array[i];

}

delete[] matrix\_array;

return det;

}

**determinant.h**

#ifndef CPP\_LAB\_4\_DETERMINANT\_H

#define CPP\_LAB\_4\_DETERMINANT\_H

double eval\_determinant(double \*\*matrix, unsigned long long int rows);

#endif //CPP\_LAB\_4\_DETERMINANT\_H

**determinant.cpp**

#include "determinant.h"

#define size\_t unsigned long long int

double \*\*sub\_matrix(double \*\*matrix, size\_t size, size\_t deletedRow, size\_t deletedCol);

double eval\_determinant(double \*\*matrix, size\_t size) {

if(size == 1) return matrix[0][0];

if(size == 2) return matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0];

double det = 0;

for(int i = 0; i < size; i++) {

double\*\* subMatrix = sub\_matrix(matrix, size, 0, i);

det += eval\_determinant(subMatrix, size - 1) \* matrix[0][i] \* (i & 1 ? -1 : 1);

for(int j = 0; j < size - 1; j++) delete[] subMatrix[j];

}

return det;

}

double \*\*sub\_matrix(double \*\*matrix, size\_t size, size\_t deletedRow, size\_t deletedCol) {

size -= 1;

double\*\* newmatrix = new double\*[size];

for(int i = 0; i < size; i++) {

newmatrix[i] = new double[size];

}

for(int row = 0, newRow = 0; row <= size; row++) {

if(row != deletedRow) {

for (int col = 0, newCol = 0; col <= size; col++) {

if (col != deletedCol) {

newmatrix[newRow][newCol] = matrix[row][col] ;

newCol++;

}

}

newRow++;

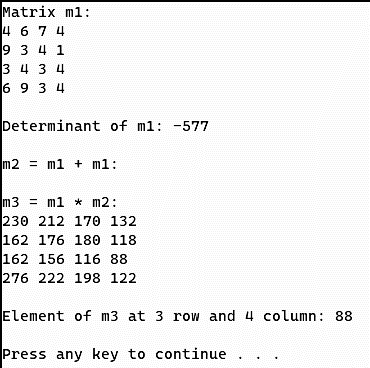
}

}

return newmatrix;

}

**Тестовые примеры**

****