МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра: Программной инженерии**

Направление подготовки: «Программная инженерия»

**Отчёт по лабораторной работе №5**

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 381908-3  Быстров Андрей Сергеевич  Проверил:  Карчков Денис Александрович |

Нижний Новгород  
2020 г.

## Содержание

1. [Описание задачи](#_Описание_задачи) 2
2. [Теория](#_Теория:) 3
3. [Методы с пояснением](#_Методы_с_пояснением) 4
4. [Методы класса Matrix](#_Методы_класса_Matrix) 10
5. [Тестирование](#_Тестирование) 16
6. [Проверка](#_Проверка) 17
7. [Вывод](#_Вывод) 18
8. [Литература](#_Литература) 19
9. [Код программы](#_Код_Vector.cpp) 20

## 

## Описание задачи

Реализовать класс Vector. Перегрузить конструкторы, деструктор, операции присваивания, скалярной суммы, скалярного произведения. Реализовать методы для определения расстояния между 2-мя векторами, косинус угла между векторами, вывод в консоль вектора и его транспонирования.

Реализовать класс Matrix. Перегрузить конструкторы, деструктор, операции присваивания, математические операции. Реализовать метод нахождения обратной матрицы, детерминанта.

## Теория:

# Вектор - это математический объект, характеризующийся величиной и направлением.

# ****Матрицами**** называются массивы элементов, представленные в виде прямоугольных таблиц, для которых определены правила математических действий. Элементами матрицы могут являться числа, алгебраические символы или математические функции.

# Применение векторов

# Вектор используется только для хранения данных. В отличии от векторов в алгебре или геометрии, в программировании, вектора могут иметь меньшее число операция или не иметь их вовсе.

## 

## Применение матриц

Матрицы, как и вектора, используются для хранения данных различного типа, но гораздо большего размера. Матрицы используются в математике для решения уравнений. Матрицы используются в программировании для создание игровых движков, для программирования трехмерной графики.

## Методы с пояснением

**Методы класса Vector**

1. Перегрузка оператора =. Метод принимает константную ссылку на объект класса Vector, если vector.Data ни на что не указывает, то программа завершается вызвав исключения VECTOR\_DATA\_NULL\_EXCEPTION. Если нет, то выполняется глубокое (поэлементное копирование. Метод возвращает \*this, так ка тип возвращаемого значения – ссылка на Vector.

Vector& Vector::operator=(const Vector& vector)

{

VectorException exception(VECTOR\_DATA\_NULL\_EXCEPTION, Exceptions::VDNEX);

if (vector.Data == NULL)

VectorExceptionHandling(exception);

Size = vector.Size;

Data = new double[Size];

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

{

Data[i] = vector.Data[i];

}

return \*this;

}

2. Перегрузка оператора +. Метод принимает константную ссылку на объект класса Vector, если размеры двух объектов не совпадают, то вызывается исключение VECTOR\_DIFFERENT\_SIZES\_EXCEPTION, если vector.Data ни на что не указывает, то вызывается исключение VECTOR\_DATA\_NULL\_EXCEPTION. Далее создается объект TEMPvector, в который записывается результат сложения векторов. Метод возвращает TEMPvector, так как тип возращаемого значения Vector.

Vector Vector::operator+(const Vector& vector) const

{

VectorException exception1(VECTOR\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::VDSEX);

VectorException exception2(VECTOR\_DATA\_NULL\_EXCEPTION, Exceptions::VDNEX);

if (Size != vector.Size)

VectorExceptionHandling(exception1);

if (vector.Data == NULL)

VectorExceptionHandling(exception2);

Vector TEMPvector;

TEMPvector.Size = Size;

TEMPvector.Data = new double[TEMPvector.Size];

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

{

TEMPvector.Data[i] = Data[i] + vector.Data[i];

}

return TEMPvector;

}

3. Перегрузка оператора -. Все тоже самое как и в перегрузке оператора -, за одним исключением – в TEMPvector записывается разность векторов.

Vector Vector::operator-(const Vector& vector) const

{

VectorException exception1(VECTOR\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::VDSEX);

VectorException exception2(VECTOR\_DATA\_NULL\_EXCEPTION, Exceptions::VDNEX);

if (Size != vector.Size)

VectorExceptionHandling(exception1);

if (vector.Data == NULL)

VectorExceptionHandling(exception2);

Vector TEMPvector;

TEMPvector.Size = Size;

TEMPvector.Data = new double[TEMPvector.Size];

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

{

TEMPvector.Data[i] = Data[i] - vector.Data[i];

}

return TEMPvector;

}

4. Перегрузка оператора \*. Данная перегрузка выполняет векторное произведение. Метод принимает один параметр – указатель на Vector – массив объектов класса Vector. В цикле проверяется все ли вектора одного массива и размер массива – 2 должен быть меньше размера вектора (пояснение: для того, чтобы посчитать векторное произведение первой строкой в матрице являются буквы ijk…, в данном случает первая строка заполняется единицами, вторая строка в матрице это вектор \*this, так как матрица должна быть квадратная, то отсюда и получается дополнительное условие). Если одно из данных не выполняется, то вызывается исключения VECTOR\_DIFFERENT\_SIZES\_EXCEPTION.

Создается объект RESULTvector, в который будет записан результат, создается два временных двумерных массива temp размера Size и tmpdata размера Size-1. Первая строка temp заполняется 1, вторая строка заполняется \*this, далле происходит копирование элементов из массива vector. tmpdata нужен для хранения матрицы без первой строки и i столбца. В цикле считается определитель tmpdata и записывается в RESULTvector.Data[i].

Далее удаляются временные массивы и метод возвращает RESULTvector.

Vector Vector::operator\*(Vector\* vector)

{

VectorException exception(VECTOR\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::VDSEX);

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

{

if(Size!=vector[i].Size)

VectorExceptionHandling(exception);

if (vector[i].ArraySize - 2 < Size)

VectorExceptionHandling(exception);

}

Vector RESULTvector(Size);

double\*\* temp = new double\* [Size];

double\*\* tmpdata = new double\* [Size-1];

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

{

tmpdata[i] = new double[Size-1];

}

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

{

temp[i] = new double[Size];

temp[0][i] = 1;

}

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

temp[1][i] = Data[i];

int k = 0;

for (int i = 2; i < Size; i++)

{

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

{

temp[i][j] = vector[k].Data[j];

}

k++;

}

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

{

Minor(temp, Size, 0, i, tmpdata);

RESULTvector.Data[i] = Determinant(tmpdata, Size - 1);

}

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

{

delete[] temp[i];

delete[] tmpdata[i];

}

delete[] temp;

temp = nullptr;

delete[] tmpdata;

tmpdata = nullptr;

return RESULTvector;

}

5. Метод для определения длины вектора, считается по формуле.

double Vector::Length() const

{

double res = 0;

if (this->Data != nullptr && this->Size > 0)

{

for (int i = 0; i < this->Size; i++)

{

res += Data[i] \* Data[i];

}

return sqrt(res);

}

else

return -1;

}

6. Метод для определения длины между векторами. Так же считается по формуле.

double Vector::LengthBetweenVectors(const Vector& vector)

{

double result = 0;

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

{

result += (vector.Data[i] - Data[i]) \* (vector.Data[i] - Data[i]);

}

return sqrt(result);

}

7. Метод для определения косинуса между векторами. Считается по формуле, использует метод Vector::Length().

double Vector::GetCosA(Vector& vector) const

{

double ResultNum = 0;

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

{

ResultNum += Data[i] \* vector.Data[i];

}

double ResultDen = this->Length() \* vector.Length();

double cos = ResultNum / ResultDen;

return cos;

}

8. Метод для транспонирования. Транспонирование вектора, в данном случает, выполняет поэлементное копирование.

void Vector::Transpose(Vector& vector)

{

VectorException exception(VECTOR\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::VDSEX);

if (this->Size != vector.Size)

VectorExceptionHandling(exception);

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

{

this->Data[i] = vector.Data[i];

}

}

## Методы класса Matrix

1. Перегрузка оператора =.

Метод принимает константную ссылку на объект класса Vector, если matrix.RowsArray ни на что не указывает, то программа завершается, вызвав исключения MATRIX\_DATA\_NULL\_EXCEPTION. Если нет, то выполняется глубокое (поэлементное копирование). Метод возвращает \*this, так ка тип возвращаемого значения – ссылка на Matrix.

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

{

MatrixException exception(MATRIX\_DATA\_NULL\_EXCEPTION, Exceptions::MDNEX);

if (matrix.RowsArray == NULL)

MatrixExceptionHandling(exception);

RowsMatrix = matrix.RowsMatrix;

ColsMatrix = matrix.ColsMatrix;

RowsArray = static\_cast<Vector\*>(operator new[](RowsMatrix \* sizeof(Vector)));

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

new (RowsArray + i) Vector(ColsMatrix);

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

{

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

{

RowsArray[i][j] = matrix.RowsArray[i][j];

}

}

return \*this;

}

2. Перегрузка оператора +. Метод принимается константную ссылку на объект класса Matrix. Если матрицы разного размера, то вызывается исключение MATRIX\_DIFFERENT\_SIZES\_EXCEPTION. Если нет, то программа продолжает выполнение. Создается объект RESULTmatrix с размером RowsMatrix x ColsMatrix. Далле выполняется поэлементное сложение и результат записывается в RESULmatrix. Метод возвращает RESULTmatrix, так как тип возвращаемого значения Matrix.

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

{

MatrixException exception(MATRIX\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::MDSEX);

if (RowsMatrix != matrix.RowsMatrix || ColsMatrix != matrix.ColsMatrix)

MatrixExceptionHandling(exception);

Matrix RESULTmatrix(RowsMatrix, ColsMatrix);

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

{

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

{

RESULTmatrix.RowsArray[i][j] = RowsArray[i][j] + matrix.RowsArray[i][j];

}

}

return RESULTmatrix;

3. Перегрузка оператора -. Работает точно так же как и перегрузка оператора +, за одним исключением, в RESULTmatrix записывается разность матриц.

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

{

MatrixException exception(MATRIX\_DIFFERENT\_SIZES\_EXCEPTION, Exceptions::MDSEX);

if (RowsMatrix != matrix.RowsMatrix || ColsMatrix != matrix.ColsMatrix)

MatrixExceptionHandling(exception);

Matrix RESULTmatrix(RowsMatrix, ColsMatrix);

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

{

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

{

RESULTmatrix.RowsArray[i][j] = RowsArray[i][j] - matrix.RowsArray[i][j];

}

}

return RESULTmatrix;

}

4. Перегрузка оператора \*. В данном случае происходит умножение матрицы на целое число. Происходит поэлементное умножение на value. Результат записывается в RESULTmatrix.

Matrix Matrix::operator\*(const int value) const

{

Matrix RESULTmatrix(RowsMatrix, ColsMatrix);

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

{

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

{

RESULTmatrix.RowsArray[i][j] = RowsArray[i][j] \* value;

}

}

return RESULTmatrix;

}

5. Перегрузка оператора \*. В данном случает происходит умножение матриц. Если ColsMatrix не равен matrix.RowsMatrix, то вызывается исключение MATRIX\_BAD\_SIZES\_EXCEPTION. Создается объект RESULTmatrix размеров RowsMatrix x matrix.ColsMatrix. Далее выполняется умножение матриц по формуле. Результат записывается в RESULTmatrix. Возвращается объект RESULTmatrix, так как тип возвращаемого значения Matrix.

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

{

MatrixException exception(MATRIX\_BAD\_SIZES\_EXCEPTION, Exceptions::MBSEX);

if (ColsMatrix != matrix.RowsMatrix)

MatrixExceptionHandling(exception);

Matrix RESULTmatrix(RowsMatrix, matrix.ColsMatrix);

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

{

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

{

for (int k = 0; k < ColsMatrix; k++)

{

RESULTmatrix.RowsArray[i][j] += RowsArray[i][k] \* matrix.RowsArray[k][j];

}

}

}

return RESULTmatrix;

}

6. Перегрузка оператора ^. Данный метод выполняется возведение матрицы в степень degree. Если степень меньше нуля, то программа завершает свое выполнение, если степень равна нуля, то матрица становится единичной. Иначе происходит возведение в

степень, используя перегрузку оператора \*. Метод возвращает RESULTvector, так как тип возвращаемого значения Matrix.

Matrix Matrix::operator^(const int degree)

{

Matrix RESULTvector(RowsMatrix, ColsMatrix);

if (degree < 0)

exit(1);

if (degree == 0)

{

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

{

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

{

RowsArray[i][j] = 1;

}

}

RESULTvector = Matrix(\*this);

return RESULTvector;

}

RESULTvector = Matrix(\*this);

for (int i = 1; i < degree; i++)

{

RESULTvector = RESULTvector \* \*this;

}

return RESULTvector;

}

7. Метод Reverse. Данный метод выполняет преобразование матрицы в обратную. Если матрица не является квадратной, то вызывается исключение NOT\_SQUARE\_EXCEPTION, если детерминант матрицы равен нуля, то вызывается исключение DIVIDED\_BY\_ZERO\_EXCEPTION, так как в дальнейшем будет происходить деление на детерминант. Находится обратная матрица, используя формулы из математики и результат записывается RESULTvector. Метод возвращается RESULTvector, так как тип возвращаемого значения Matrix.

Matrix Matrix::Reverse()

{

MatrixException exception1(NOT\_SQUARE\_EXCEPTION, Exceptions::NSQEX);

MatrixException exception2(DIVIDED\_BY\_ZERO\_EXCEPTION, Exceptions::DBZEX);

if (RowsMatrix != ColsMatrix)

MatrixExceptionHandling(exception1);

Matrix RESULTmatrix(RowsMatrix, RowsMatrix);

double det = this->Determinant();

if (det == 0)

MatrixExceptionHandling(exception2);

Matrix TEMPmatrix(RowsMatrix - 1, RowsMatrix - 1);

int degree;

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

{

degree = i % 2 == 0 ? 1 : -1;

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

{

TEMPmatrix = Minor(i, j);

RESULTmatrix.RowsArray[i][j] = degree \* TEMPmatrix.Determinant() / det;

degree = -degree;

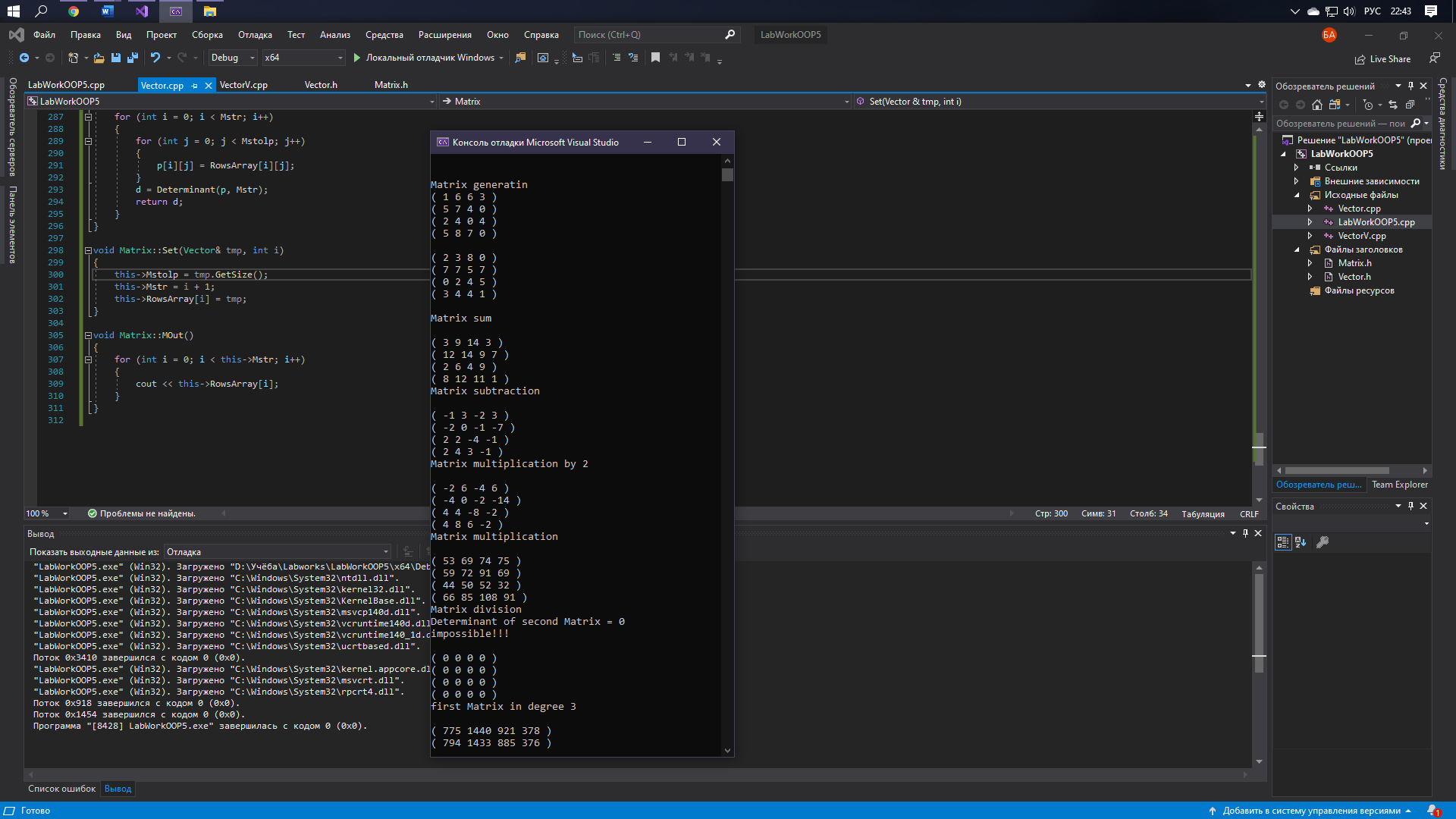
}

}

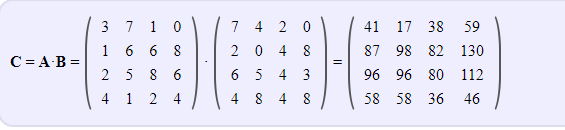
return RESULTmatrix;

## Тестирование

# 



## Проверка



## Вывод

В данной лабораторной работе я освоил создание массива объектов класса. Так же научился использовать свои математические знания для создания методов для нахождения результата операций для векторов и матриц.

## Литература

С.Н. Карпенко Основы программирования на языке C++

<https://ravesli.com/uroki-cpp/>

<https://habr.com/ru/post/489666/>

## Код Vector.cpp

#include "Matrix.h"

void GetMatr(double\*\* mas, double\*\* p, int i, int j, int m)

{

int ki, kj, iSmesh, jSmesh;

iSmesh = 0;

for (ki = 0; ki < m - 1; ki++) {

if (ki == i)

iSmesh = 1;

jSmesh = 0;

for (kj = 0; kj < m - 1; kj++) {

if (kj == j)

jSmesh = 1;

p[ki][kj] = mas[ki + iSmesh][kj + jSmesh];

}

}

}

int Determinant(double\*\* mas, int m)

{

int i, j, k, n;

double\*\* p, d;

p = new double\* [m];

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

p[i] = new double[m];

j = 0; d = 0;

k = 1; //(-1) в степени i

n = m - 1;

if (m < 1) cout << "Impossible";

if (m == 1) {

d = mas[0][0];

return(d);

}

if (m == 2) {

d = mas[0][0] \* mas[1][1] -(mas[1][0] \* mas[0][1]);

return(d);

}

if (m > 2) {

for (i = 0; i < m; i++) {

GetMatr(mas, p, 0,i, m);

d = d + k \* mas[0][i] \* Determinant(p, n);

k = -k;

}

}

return(d);

}

Matrix::Matrix()

{

this->Mstr = 3;

this->Mstolp = 3;

this->RowsArray = static\_cast<Vector\*>(operator new[](this->Mstr \* sizeof(Vector)));

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

{

new (RowsArray + i) Vector(Mstolp);

}

}

Matrix::~Matrix()

{

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

{

RowsArray[i].~Vector();

}

this->Mstr = 0;

this->Mstolp = 0;

operator delete[] (RowsArray);

}

Matrix::Matrix(int str, int stolp)

{

this->Mstr = str;

this->Mstolp = stolp;

this->RowsArray = static\_cast<Vector\*>(operator new[](this->Mstr \* sizeof(Vector)));

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

{

new (RowsArray + i) Vector(Mstolp);

}

}

Matrix::Matrix(const Matrix& tmp)

{

if (tmp.RowsArray == nullptr)

throw 1;

this->Mstr = tmp.Mstr;

this->Mstolp = tmp.Mstolp;

this->RowsArray = static\_cast<Vector\*>(operator new[](this->Mstr \* sizeof(Vector)));

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

{

new (RowsArray + i) Vector(Mstolp);

}

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

{

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

{

RowsArray[i][j] = tmp.RowsArray[i][j];

}

}

}

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

{

if (tmp.RowsArray == nullptr)

throw 1;

this->Mstr = tmp.Mstr;

this->Mstolp = tmp.Mstolp;

this->RowsArray = static\_cast<Vector\*>(operator new[](this->Mstr \* sizeof(Vector)));

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

{

new (RowsArray + i) Vector(Mstolp);

}

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

{

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

{

RowsArray[i][j] = tmp.RowsArray[i][j];

}

}

return \*this;

}

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

{

Matrix res(tmp.Mstr,tmp.Mstolp);

//double\*\* resRow = new double\*[Mstr];

if (Mstr != tmp.Mstr)

{

cout << "m or n no equal"<<endl;

return res;

}

if (Mstolp != tmp.Mstolp)

{

cout << "m or n no equal" << endl;

return res;

}

else {

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

{

res.RowsArray[i] = this->RowsArray[i] + tmp.RowsArray[i];

}

}

return res;

}

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

{

Matrix res(tmp.Mstr, tmp.Mstolp);

//double\*\* resRow = new double\*[Mstr];

if (Mstr != tmp.Mstr)

{

cout << "m or n no equal" << endl;

return res;

}

if (Mstolp != tmp.Mstolp)

{

cout << "m or n no equal" << endl;

return res;

}

else {

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

{

res.RowsArray[i] = this->RowsArray[i] - tmp.RowsArray[i];

}

}

return res;

}

Matrix Matrix::operator\*(const int& k) const

{

Matrix res(Mstr, Mstolp);

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

{

res.RowsArray[i] = RowsArray[i] \* k;

}

return res;

}

Matrix Matrix::operator/(const int& k) const

{

Matrix res(Mstr, Mstolp);

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

{

res.RowsArray[i] = RowsArray[i] / k;

}

return res;

}

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

{

Matrix res(Mstr,tmp.Mstolp);

int ki = 0,kj=0,k ;

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

{

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

{

for (int k = 0; k <tmp.Mstolp ; k++)

{

res.RowsArray[i][j] +=(this->RowsArray[i][k] \* tmp.RowsArray[k][j]);

}

}

}

return res;

}

Matrix Matrix::operator/(const Matrix& tmp) const

{

Matrix res(Mstr, Mstolp),j(Mstr, Mstolp);

int d;

res = tmp;

d = res.D();

cout << "Determinant of second Matrix = " << d<<endl;

if (d == 0)

{

cout << "impossible!!!" << endl;

return j;

}

res = res.T();

res = res / d;

cout << "Invert matrix " << endl;

res.MOut();

res = res\* \*this;

return res;

}

Matrix Matrix::operator^(const int tmp) const

{

Matrix res(Mstr, Mstolp);

res = \*this;

if (tmp > 1)

{

for (int i = 1; i < tmp; i++)

{

res = res \* \*this;

}

return res;

}

else if (tmp == 0)

{

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

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

{

res.RowsArray[i][j] = 1;

}

return res;

}

else

{

cout << "impossible!!!" << endl;

return \*this;

}

}

Matrix Matrix::T()

{

Matrix tmp(Mstolp, Mstr);

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

{

for (int j = i; j < Mstr; ++j)

{

tmp.RowsArray[i][j] = this->RowsArray[i][j];

this->RowsArray[i][j] = this->RowsArray[j][i];

this->RowsArray[j][i] = tmp.RowsArray[i][j];

}

}

return \*this;

}

int Matrix::D()

{

int d;

double\*\* p;

p = new double\* [Mstolp];

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

p[i] = new double[Mstolp];

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

{

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

{

p[i][j] = RowsArray[i][j];

}

d = Determinant(p,Mstr);

return d;

}

void Matrix::Set( Vector& tmp, int i)

{

this->Mstolp = tmp.GetSize();

this->Mstr = i+1;

this->RowsArray[i] = tmp;

}

void Matrix::MOut()

{

for (int i = 0; i < this->Mstr; i++)

{

cout << this->RowsArray[i];

}

}

# 

## Код Matrix.h

#include"Vector.h";

class Matrix

{

private:

int Mstr;

int Mstolp;

Vector\* RowsArray;

public:

Matrix();

~Matrix();

Matrix(int str, int stolp);

Matrix(const Matrix& tmp);

Matrix& operator=(const Matrix& tmp);

Matrix operator+(const Matrix& tmp)const;

Matrix operator-(const Matrix& tmp)const;

Matrix operator\*(const int& k)const;

Matrix operator/(const int& k)const;

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

Matrix operator/(const Matrix& tmp)const;

Matrix operator^(const int tmp) const;

Matrix T();

int D();

void Set( Vector& tmp, int i);

void MOut();

};

## Код VectorV.cpp

#include "Vector.h"

#include<math.h>

int det = 0, step = -1;

void Minor(double\*\* mas, double\*\* p, int i, int j, int m)

{

int ki, kj, iSmesh, jSmesh;

iSmesh = 0;

for (ki = 0; ki < m - 1; ki++) {

if (ki == i)

iSmesh = 1;

jSmesh = 0;

for (kj = 0; kj < m - 1; kj++) {

if (kj == j)

jSmesh = 1;

p[ki][kj] = mas[ki+ iSmesh][kj + jSmesh];

}

}

}

void PrintMatr(double\*\* mas, int m) {

int i, j;

for (i = 0; i < m; i++) {

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

cout << mas[i][j] << " ";

cout << endl;

}

}

double Determinant(double\*\* mas, int m,double \*dres,int ii,bool three) {

int j, k, n;

double d;

double\*\* p;

p = new double\* [m];

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

p[i] = new double[m];

j = 0; d = 0;

k = 1;

n = m - 1;

if (m < 1)

cout << "ne robit";

if (m == 1)

{

d = mas[0][0];

return(d);

}

if (m == 2) {

d = mas[0][0] \* mas[1][1] -(mas[1][0] \* mas[0][1]);

if(three == true)

dres[ii] = d;

return(d);

}

if (m > 2) {

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

Minor(mas, p, 0, i, m);

d = d + (k \* mas[0][i] \* Determinant(p, n,dres,i, three));

if (three == false)

dres[ii] = d;

k = -k;

}

}

return(d);

}

Vector Vector::VecProduct(Vector\* V)

{

bool \_3;

if (Size == 3)

\_3 = true;

else

\_3 = false;

double\*\* p;

p = new double\* [Size];

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

p[i] = new double[Size];

double\* mRes = new double[Size];

Vector res(Size);

double d;

double\*\* m = new double\* [Size];

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

m[i] = new double[Size];

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

{

m[0][i] = 1;

}

for (int i = 1; i < Size; i++)

{

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

{

m[i][j] = V[i].Data[j];

}

}

Minor(m, p, 0, 0, Size);

Determinant(p, Size-1, mRes, 0,\_3);

d = mRes[0];

Determinant(m, Size, mRes, 0,\_3);

for (int i = 1; i < Size; i++)

{

res.Data[0] = d;

res.Data[i] = mRes[i];

}

return res;

}

void Vector::Random()

{

for (int i = 0; i < this->Size; i++)

{

this->Data[i] = rand()%9;

}

}

int Vector::GetSize()

{

return this->Size;

}

Vector::Vector()

{

Data = new double[3];

Size = 3;

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

{

Data[i] = 0;

}

//cout << "simple constr " << this << endl;

}

//

//Vector::Vector(int \_size, double\* \_data)

//{

// if (\_data == NULL)

// throw std::exception();

// this->Size = \_size;

// for (int i = 0; i < \_size; i++)

// {

// \*(\_data +i)= 0;

// }

// cout << "full constr " << this << endl;

//

//}

Vector::Vector(int \_size)

{

this->Size = \_size;

this->Data = new double[\_size];

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

{

this->Data[i] = 0;

}

//cout <<"constr size "<< this << endl;

}

Vector::Vector(const Vector& v)

{

if (v.Data == NULL)

throw std::exception("invalid data");

Size = v.Size;

Data = new double[Size];

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

{

Data[i] = v.Data[i];

}

}

Vector::~Vector()

{

delete[] this->Data;

this->Data = nullptr;

this->Size = 0;

//cout <<"destr "<< this<<endl;

}

Vector& Vector::operator=(const Vector& v)

{

Size = v.Size;

this->Data = new double[Size];

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

{

this->Data[i] = v.Data[i];

}

return \*this;

}

Vector Vector::operator+(const Vector& tmp) const

{

int s = this->Size;

if (this->Size > tmp.Size)

{

s = this->Size;

Vector ttp(s);

for (int i = 0; i < tmp.Size; i++)

{

ttp.Data[i] = tmp.Data[i];

}

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = this->Data[i] + ttp.Data[i];

return sum;

}

else if (this->Size < tmp.Size)

{

s = tmp.Size;

Vector ttp(s);

for (int i = 0; i < this->Size; i++)

{

ttp.Data[i] = this->Data[i];

}

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = tmp.Data[i] + ttp.Data[i];

return sum;

}

else

{

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = tmp.Data[i] + this->Data[i];

return sum;

}

}

Vector Vector::operator-(const Vector& tmp) const

{

int s = this->Size;

if (this->Size > tmp.Size)

{

s = this->Size;

Vector ttp(s);

for (int i = 0; i < tmp.Size; i++)

{

ttp.Data[i] = tmp.Data[i];

}

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = this->Data[i] - ttp.Data[i];

return sum;

}

else if (this->Size < tmp.Size)

{

s = tmp.Size;

Vector ttp(s);

for (int i = 0; i < this->Size; i++)

{

ttp.Data[i] = this->Data[i];

}

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = ttp.Data[i] - tmp.Data[i];

return sum;

}

else

{

Vector sum(s);

for (int i = 0; i < sum.Size; i++)

sum.Data[i] = this->Data[i] - tmp.Data[i];

return sum;

}

}

double Vector::operator\*(const Vector& tmp) const

{

int s = Size;

double res = 0;

if (this->Size > tmp.Size)

{

s = this->Size;

Vector ttp(s);

for (int i = 0; i < tmp.Size; i++)

{

ttp.Data[i] = tmp.Data[i];

}

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

res += this->Data[i] \* ttp.Data[i];

return res;

}

else if (this->Size < tmp.Size)

{

s = tmp.Size;

Vector ttp(s);

for (int i = 0; i < this->Size; i++)

{

ttp.Data[i] = this->Data[i];

}

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

res += tmp.Data[i] \* ttp.Data[i];

return res;

}

else

{

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

res+= tmp.Data[i] \* this->Data[i];

return res;

}

}

Vector Vector::operator\*(const int tmp) const

{

Vector res(Size);

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

{

res.Data[i] = this->Data[i] \* tmp;

}

return res;

}

Vector Vector::operator/(const int tmp) const

{

Vector res(Size);

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

{

res.Data[i] = this->Data[i] / tmp;

}

return res;

}

double& Vector::operator[](const int tmp)

{

return this->Data[tmp];

}

double Vector::lenght()

{

double res = 0;

if (this->Data != nullptr && this->Size > 0)

{

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

res += Data[i] \* Data[i];

return sqrt(res);

}

else

return -1;

}

double Vector::lenghtbetweenvectors(const Vector& tmp)

{

double res = 0;

if (this->Data != nullptr && this->Size > 0)

{

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

res += (tmp.Data[i] - Data[i]) \* (tmp.Data[i] - Data[i]);

return sqrt(res);

}

else

return -1;

}

double Vector::Cos(Vector& tmp)

{

double mult = 0, res1, res;

//int s = this->Size;

/\*if (this->Size > tmp.Size)

{

s = this->Size;

Vector ttp(s);

for (int i = 0; i < tmp.Size; i++)

{

ttp.Data[i] = tmp.Data[i];

}

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

{

mult += this->Data[i] \* ttp.Data[i];

}

}

else if (this->Size < tmp.Size)

{

s = tmp.Size;

Vector ttp(s);

for (int i = 0; i < this->Size; i++)

{

ttp.Data[i] = this->Data[i];

}

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

{

mult += tmp.Data[i] \* ttp.Data[i];

}

}

else

{

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

{

mult += this->Data[i] \* tmp.Data[i];

}

}\*/

mult = \*this \* tmp;

res = this->lenght();

res1 = tmp.lenght();

double coss = mult / (res \* res1);

return coss;

}

std::istream& operator>>(std::istream& in, const Vector& tmp)

{

for (int i = 0; i < tmp.Size; i++)

{

cout << "Input "<<i+1<<"-s vector coordinate"<<endl;

in >> tmp.Data[i];

}

return in;

}

std::ostream& operator<<(std::ostream& out, const Vector& tmp)

{

out << "( ";

for (int i = 0; i < tmp.Size; i++)

{

out << tmp.Data[i]<<" ";

}

out << ")"<<endl;

return out;

}

## Код Vector.h

#include <iostream>

#include <math.h>

#include <stdlib.h>

#include <time.h>

using namespace std;

class Vector

{

public:

Vector();

//Vector(int \_size, double\*\_data);

Vector(int \_size);

Vector(const Vector& v);

~Vector();

Vector& operator=(const Vector& tmp);

Vector operator+(const Vector& tmp) const;

Vector operator-(const Vector& tmp)const;

double operator\*(const Vector& tmp)const;

Vector operator\*(const int tmp)const;

Vector operator/(const int tmp)const;

double& operator[](const int tmp);

double lenght();

double lenghtbetweenvectors(const Vector &tmp);

double Cos(Vector& tmp);

friend std::istream& operator>>(std::istream& in, const Vector& tmp);

friend std::ostream& operator<<(std::ostream& out, const Vector& tmp);

Vector VecProduct(Vector\* Vec);

void Random();

int GetSize();

private:

int Size;

double\* Data;

};

**Код LabWorkOOP5.cpp**

#include "Matrix.h"

#include <locale>

int main()

{

setlocale(LC\_ALL,"Rus");

srand((unsigned int)time(0));

int n;

cout << "Vector size"<<endl;

do {

cin >> n;

} while (n < 3);

Vector v1(n), v2(n), sum(n), resVec(n);

double res;

cout << "Vector 1"<< endl;

cin >> v1;

cout << "Vector 2" << endl;

cin >> v2;

Vector\* m = new Vector[n];

cout<<endl<<"Lenght between vectors "<<v1.lenghtbetweenvectors(v2)<<endl<<endl;

sum = v1 + v2;

cout <<"Sum result "<< sum << endl;

sum = v1 - v2;

cout <<"Subtraction result " << sum << endl;

res = v1 \* v2;

cout <<"Inner product result "<< res<<endl;

m[2] = v2;

m[1] = v1;

for (int i = 3; i < n; i++)

{

v2.Random();

m[i] = v2;

}

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

// //mas[i] = new int[m];

// for (int j = 0; j < n; j++) {

// cout << "main mas[" << i << "] [" << j << "]= " << jopa[i][j] << endl;

// }

//}

resVec = v1.VecProduct(m);

cout <<endl<< "Vector Mult result " << resVec<<endl;

Matrix m1(n,n), m2(n,n),m3;

Matrix ResM(n, n);

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

{

v1.Random();

v2.Random();

m1.Set(v1, i);

m2.Set(v2, i);

}

cout << endl;

cout << "Matrix generatin"<<endl;

m1.MOut();

cout << endl;

m2.MOut();

cout << endl;

cout << "Matrix sum" << endl;

ResM = m1+m2;

cout << endl;

ResM.MOut();

cout << "Matrix subtraction" << endl;

ResM = m1 - m2;

cout << endl;

ResM.MOut();

cout << "Matrix multiplication by 2" << endl;

ResM = ResM \* 2;

cout << endl;

ResM.MOut();

cout << "Matrix multiplication" << endl;

ResM = m1 \* m2;

cout << endl;

ResM.MOut();

cout << "Matrix division" << endl;

ResM = m1 / m2;

cout << endl;

ResM.MOut();

cout << "first Matrix in degree 3" << endl;

ResM = m1^3;

cout << endl;

ResM.MOut();

}