Assignment 2, Nov.15.2018 **Due Date: Dec.13.2018**

Name: Kadircan KURTULUŞ
Number: 16015001
Course: KOM3191 Object-Oriented Programming
Date: 29 November

1. Using the header file **Matrix.h** (check the course webpage) type the implementation file **Matrix.cpp**

   Hint: if you cannot find the relation between the `float* data` pointer and the matrix form, use the following empty constructor function

```
Matrix::Matrix()
// initialize Matrix class object with rowN=1, colN=1, and a zero value
{
    rowN=1;
    colN=1;
    data=new float[rowN*colN];

    for (int i=0; i<rowN; i++)
        for (int j=0; j<colN; j++)
            data[i*rowN+j]=0;
}
```

2. Submit your assignment

Dr Muharrem Mercimek

a) Complete and submit your assignment yourself.
b) The due date is firm and assignment can be submitted by the **end of this date.** “NO OTHER EXCEPTION”
c) Print out your document and hand it in.

```cpp
#include <iostream>
#include "Matrix.h"
using namespace std;
Matrix::Matrix()
{
    rowN = 1;
    colN = 1;
    data = new float[1];
    *data = 0;
}
Matrix::Matrix(const int rN, const int cN)
{
    rowN = rN;
    colN = cN;
    data = new float[rowN*colN];
    for (int i = 0; i < rowN; i++)
        for (int j = 0; j < colN; j++)
            data[i*colN + j] = 0;
}
Matrix::Matrix(const Matrix &srcMatrix)
{
    rowN = srcMatrix.rowN;
    colN = srcMatrix.colN;
    data = srcMatrix.data;
}
Matrix::Matrix(const int rN, const int cN, const float const *srcPtr)
{
    rowN = rN;
    colN = cN;
    data = new float[rowN*colN];
    for (int i = 0; i < rowN; i++)
        for (int j = 0; j < colN; j++)
            data[i*colN + j] = *(srcPtr++);
}
const float* Matrix::getData()const
{
    return data;
}
int Matrix::getRowN()const
{
    return rowN;
}
int Matrix::getColN()const
{
    return colN;
}
void Matrix::print()const
{
    for (int i = 0; i < rowN; i++)
    {
        for (int j = 0; j < colN; j++)
            cout << data[i*colN + j] << ' ';
        cout << endl;
    }
}
Matrix Matrix::transpose() const
```

```cpp
{
    Matrix temp(colN, rowN);
    for (int i = 0; i < colN; i++)
        for (int j = 0; j < rowN; j++)
            temp.data[i*rowN + j] = data[i + j*colN];
    return temp;
}
Matrix Matrix::operator+(const Matrix &rhsMatrix)const
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            Matrix temp(rowN, colN);
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    temp.data[i*colN + j] = data[i*colN + j] + rhsMatrix.data
                    [i*colN + j];
            return temp;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
Matrix Matrix::operator-(const Matrix &rhsMatrix)const
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            Matrix temp(rowN, colN);
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    temp.data[i*colN + j] = data[i*colN + j] - rhsMatrix.data
                    [i*colN + j];
            return temp;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
Matrix Matrix::operator*(const Matrix &rhsMatrix)const
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            Matrix temp(rowN, colN);
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    temp.data[i*colN + j] = data[i*colN + j] * rhsMatrix.data[i*colN + j];
            return temp;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
float Matrix::operator()(const int r, const int c)const
{
    try
    {
        if (r <= 0)
            throw invalid_argument("Index in position 1 is invalid. Array indices must be positive integers.");
        else if (r - 1 < rowN)
        {
            if (c <= 0)
                throw invalid_argument("Index in position 2 is invalid. Array indices must be positive integers.");
            else if (c - 1 < colN)
                return data[(r - 1)*colN + c - 1];
            else
                throw out_of_range("Index in position 2 exceeds array bounds.");
        }
        else
            throw out_of_range("Index in position 1 exceeds array bounds.");
    }
    catch (const exception &ex)
    {
        cerr << ex.what() << endl;
    }
}
Matrix& Matrix::operator=(const Matrix &rhsMatrix)
{
    rowN = rhsMatrix.rowN;
    colN = rhsMatrix.colN;
    data = rhsMatrix.data;
    return *this;
}
Matrix& Matrix::operator+=(const Matrix &rhsMatrix)
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    data[i*colN + j] += rhsMatrix.data[i*colN + j];
            return *this;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
Matrix& Matrix::operator-=(const Matrix &rhsMatrix)
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    data[i*colN + j] -= rhsMatrix.data[i*colN + j];
            return *this;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
Matrix& Matrix::operator*=(const Matrix &rhsMatrix)
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    data[i*colN + j] *= rhsMatrix.data[i*colN + j];
            return *this;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
int Matrix::operator==(const Matrix &rhsMatrix)const
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
```

```
                    if (data[i*colN + j] != rhsMatrix.data[i*colN + j])
                        return 0;
            return 1;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
int Matrix::operator!=(const Matrix &rhsMatrix)const
{
    try
    {
        if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
        {
            for (int i = 0; i < rowN; i++)
                for (int j = 0; j < colN; j++)
                    if (data[i*colN + j] != rhsMatrix.data[i*colN + j])
                        return 1;
            return 0;
        }
        else
            throw logic_error("Matrix dimensions must agree.");
    }
    catch (const logic_error &ex)
    {
        cerr << ex.what() << endl;
    }
}
```