Øvelse 7 – Matrix

Opgaven går ud på at skrive en klasse til at repræsentere en matrix. Formålet med opgaven er:

- at blive fortrolig med references, smart pointers og operator overload
- at træne løsing af opgaver der involverer std::vector

Til at håndtere lagring af data i matricen skal I benytte en std::vector:

```
private std::vector<double> data
```

Desuden skal der være to private variable der indeholder størelsen på matricen:

```
public int rows
public int cols
```

Programmér klassen ud fra følgende specifikaiton.

Beskrivelse af public memberfunktionerne (metoderne)

`Matrix(int rows, int cols, double val = 0)`
Constructor som initialiserer matricen med val i hvert index (default er 0) og initialisere feltvariablene rows og cols (se ovenfor).

`int getRows() og int getCols()`
Getter metoder til rows og cols (data er den interne data repræsentation af matricen, så den vil vi ikke lave getter metode til)

`void print()`
Printer matricen vha. cout, dvs. Skriver alle værdierne I terminalen så man kan se hvad matricen indeholder

`double read(int r, int c)`
returnerer værdien af et enkelt entry i matricen.

`void set(int r, int c, double val)`
Sætter index (r,c) til at have værdien val. I skal benytte 0-indexering.

`double& at(int r, int c)`
returnerer en reference til en enkelt entry i matricen.

`Matrix add(Matrix& matrix)`

Tjekker at matricen selv (A) og input matrix (B) har samme størrelse og laver elementvis addition (A+B). Returnerer resultatet som en ny matrix.

`Matrix multiply(Matrix& matrix)`
Returnerer matrix produktet mellem den matrice som funktionen bliver kaldt på og input matricen. Dvs. hvis matricen som denne metode bliver kaldt på hedder A og input matricen hedder B, så returnerer denne funktion AB. Husk at tjekke at størrelserne er kompatible, dvs. hvis A: ma x na og B: mb x nb, så skal na == mb og produktet AB bliver ma x nb.

`void transpose()`
Transponerer matricen.

`std::shared_ptr<Matrix> add(std::shared_ptr<Matrix> matrix)`
Magen til den anden add metode, men benytter smart pointers

`std::shared_ptr<Matrix> multiply(std::shared_ptr<Matrix> matrix)`
Magen til den anden multiply metode, men benytter smart pointers

`double& operator()(int r, int)`
overloader parentes operatoren så den kalder at(r,c) metoden

`Matrix operator+(Matrix m)`
overloader plus operatoren så den kalder add() metoden

`Matrix operator*(Matrix m)`
overloader gange operatoren så den kalder multiply() metoden

Jeres klasse skal kunne køres med følgende main.cpp fil (som også er udleveret på itsLearning):

```
#include <iostream>
#include "matrix.h"
#include <math.h>
#include <cmath>
#include <sstream>
#include <string>

void setMatrixValues(Matrix& m, std::vector<double> v){
    for(int r=0; r<m.getRows(); r++){
        for(int c=0; c<m.getCols(); c++){
            m.at(r,c) = v[r*m.getCols() + c];
        }
    }
}

std::string mat2str(Matrix m, double eps=1e-9){
     std::stringstream ss;
     ss.precision(3); //compare results down to three decima places
    for(int r=0; r<m.getRows(); r++){
        for(int c=0; c<m.getCols(); c++){
            if (m(r,c)<eps && m(r,c)>-eps) ss << 0; //allow small numerical
deviations from zero
            else ss << m(r,c);
            if(c<m.getCols()-1){
                ss << " ";
            }
        }
        ss << ";";
        if(r<m.getRows()-1){
            ss<<" ";
        }
    }
    return ss.str();
}

void test(Matrix test, std::string target, std::string msg){
    std::string mStr = mat2str(test);
    std::string equal = mStr==target ? "Success: " : "Failed:  ";
    std::cout << equal << msg << "\t " << mStr << " " << target << std::endl;
}

template <class T1>
void test(T1 test, T1 target, std::string msg){
    std::string equal = test==target ? "Success: " : "Failed:  ";
    std::cout << equal << msg << "\t " << test << " " << target << std::endl;
}

void testEps(double test, double target, double eps, std::string msg){
    std::string equal = std::pow(std::pow(target-test,2.0),0.5) < eps ?
"Success: " : "Failed:  ";
    std::cout << equal << msg << "\t " << test << " " << target << std::endl;
```

```
}

int main(){
    //Construction and index access
    Matrix mat(3,4);
    test(mat,"0 0 0 0; 0 0 0 0; 0 0 0 0;", "simple constructor");
    Matrix mat2(3,4,3);
    test<double>(mat2.at(0,0),3,"at method");
    test<double>(mat2(2,3),3,"() operator");

    //Transpose
    setMatrixValues(mat,std::vector<double>{0,1,2,3,4,5,6,7,8,9,10,11});
    Matrix mat3(mat); //Make a copy of mat
    test(mat3,"0 1 2 3; 4 5 6 7; 8 9 10 11;", "creation of mat3");
    mat3.transpose();
    test(mat3,"0 4 8; 1 5 9; 2 6 10; 3 7 11;", "transpose()");

    //Addition
    std::shared_ptr<Matrix> ptr1 = std::make_shared<Matrix>(4,3);
    setMatrixValues(*ptr1,std::vector<double>{0,0,1,1,1,2,2,2,3,3,3,4});
    test(*ptr1,"0 0 1; 1 1 2; 2 2 3; 3 3 4;", "Creation of mat4");
    Matrix mat1 = *ptr1;
    Matrix mat6 = ptr1->add(mat3);
    test(mat6,"0 4 9; 2 6 11; 4 8 13; 6 10 15;", "add() refencence version");
    std::shared_ptr<Matrix> ptr2 = mat3.add(ptr1);
    test(*ptr2,"0 4 9; 2 6 11; 4 8 13; 6 10 15;", "add() shared_ptr version");
    ptr1->transpose();
    Matrix mat7 = ptr1->multiply(mat6);
    test(mat7,"28 52 82; 28 52 82; 40 80 130;","multiply() reference version");
    std::shared_ptr<Matrix> mat9 = mat6.multiply(ptr1);
    test(*mat9,"9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108;","multiply()
shared_ptr version");

    //Inner product
    test(Matrix(1,3,2)*Matrix(3,1,2),"12;","inner product");

    //outer product
    test(Matrix(3,1,2)*Matrix(1,3,2),"4 4 4; 4 4 4; 4 4 4;","outer product");

    // + and * operators
    Matrix mat10 = mat3+mat1;
    test(mat10,"0 4 9; 2 6 11; 4 8 13; 6 10 15;","+ operator");
    mat1.transpose();
    Matrix mat8 = mat6*mat1;
    test(mat8,"9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108;","* operator
(matrix multiplication)");

}
```