Pointgivende aktivitet nummer 2

Denne opgaves tema er:

- Referencer og smart pointers
- std::vector som underliggende datastruktur
- Nedarvning
- Operator overload

- Mulige antal point for udførsel af pointgivende aktivitet: 3

- **Deadline: tirsdag den 30. april kl. 23.59**
  - Aflevering på itsLearning (Jeg opretter et topic)

- Denne pointgivende aktivitet består af to opgaver, hvor der skal programmeres klasserne Matrix og RotMat2D. Efter løsning af opgave 1 og 2 skal nedenstående main metode kunne køres (main.cpp er vedlagt opgaven). Som svar til opgaverne skal I aflevere én zip-fil indeholdende følgende filer:
  - **output.pdf** – Screenshot af outputtet af jeres program. Hvis det kun er en begrænset del af programmet der kan compile, skal I udkommentere den del der ikke virker og tage et screen-shot af det I har udkommenteret, samt outputtet af den del af programmet som compiler og kører som det skal.
  - **matrix.h**
  - **rotmat.h**
  - **matrix.cpp** (hvis I har delt klassen op I h- og cpp-filer)
  - **rotmat.cpp** (hvis I har delt klassen op I h- og cpp-filer)

Opgave 1 – Matrix

Den første (og største) opgave er at løse opgave ex07_matrix fra lektion 7. Efter løsning af opgaven skal I lave følgende lille ændring:

- Tilføj virtual keyworded til metoderne print() og transpose()

Opgave 2 – RotMat2D

Denne opgave går ud på at skrive en klasse til at repræsentere en rotationsmatrix til rotationer i 2D. En rotationmatrix har følgende form:

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Vær opmærksom på at denne opgave udelukkende benytter radianer til at beskrive vinkler. Da en rotations matrix er en normal matrix med yderligere egenskaber skal I benytte nedarvning til at genbruge funktionaliteten fra Opgave 1. Opgaven går altså ud på at skrive klassen RotMat2D som beskrevet nedenfor:

1. Opret rotmat.h og evt. rotmat.cpp hvis I vælger at dele klassen op i en source og en header fil.
2. RotMat2D skal nedarve fra Matrix og har én privat variabel, angle, som angiver rotationen (med fortegn og angivet i radianer) som rotationsmatricen repræsenterer. Alle nedenstående metoder er public.
3. Skriv en default constructor RotMat2D() som kalder super-klassens konstructor så den bliver initialiseret som en 2x2 matrix. Initialiser matricen til identitetsmatricen (dvs. [1 0; 0 1]). Initialiser angle til at være 0 radianer.
4. Skriv constructoren RotMat2D(double angle) som tager en vinkel i radianer som input og initialiserer rotations matrix ud fra denne vinkel. Se definitionen af en rotationsmatrix ovenfor.
   - Hint: I skal kalde superklassens konstruktor så den bliver initialiseret med de korrekte værdier
5. Skriv klassen print() som overskriver print metoden i super-klassen. Husk at angive metoden med override key-wordet. Metoden skal kalde super-klassens print metode som printer matricen til terminalen, men derover skal den også printe den vinkel som metoden representerer.
   - Et eksempel på formatet af print metodens output kan ses på outputet af main metoden som er vist nedenfor.
6. Skriv klassen transpose() som overskriver transpose metoden i super-klassen. Husk at angive metoden med override key-wordet. Metoden skal kalde super-klassens transpose metode, men derudover skal den også ændre angle til at have omvendt fortegn (Den trasponerede af en rotationsmatrix er dens inverse, dvs. det er en rotation om samme akse men med omvendt fortegn.)

main.cpp (ligger på itsLearning)

```cpp
#include <iostream>
#include "matrix.h"
#include "rotmat.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)");


    //Rotation matrices
    RotMat2D identity;
    test(identity,"1 0; 0 1;", "Identity");
    RotMat2D ninety(M_PI/2);
    test(ninety,"0 -1; 1 0;", "ninety");
    ninety.transpose();
    test(ninety,"0 1; -1 0;", "ninety trasposed");
    RotMat2D thirty(30*M_PI/180);
    test(thirty,"0.866 -0.5; 0.5 0.866;", "thirty");
    RotMat2D sixty(60*M_PI/180);
    test(sixty,"0.5 -0.866; 0.866 0.5;", "sixty");
    Matrix allThree = thirty * sixty;
    test(allThree,"0 -1; 1 0;", "thirty*sixty");
    allThree = allThree * ninety;
    test(allThree,"1 0; 0 1;", "thirty*sixty* ninety (ninety is transposed)");


    //Rotate v by thirty degrees
    Matrix v(2,1,0);
    v(0,0) = 2;
    v(1,0) = 3;
    test(thirty*v,"0.232; 3.6;", "Rotation of vector using rotation matrix");


    //Test correct override of virtual methods print and transpose
    std::cout << std::endl;
    Matrix * p = &thirty;
    p->print();
    p->transpose();
    p->print();


}
```

Output af main.cpp

```
Success: simple constructor                0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0;
Success: at method     3 3
Success: () operator   3 3
Success: creation of mat3                  0 1 2 3; 4 5 6 7; 8 9 10 11; 0 1 2 3; 4 5 6 7; 8 9 10 11;
Success: transpose()   0 4 8; 1 5 9; 2 6 10; 3 7 11; 0 4 8; 1 5 9; 2 6 10; 3 7 11;
Success: Creation of mat4                  0 0 1; 1 1 2; 2 2 3; 3 3 4; 0 0 1; 1 1 2; 2 2 3; 3 3 4;
Success: add() refencence version          0 4 9; 2 6 11; 4 8 13; 6 10 15; 0 4 9; 2 6 11; 4 8 13; 6 10 15;
Success: add() shared_ptr version          0 4 9; 2 6 11; 4 8 13; 6 10 15; 0 4 9; 2 6 11; 4 8 13; 6 10 15;
Success: multiply() reference version      28 52 82; 28 52 82; 40 80 130; 28 52 82; 28 52 82; 40 80 130;
Success: multiply() shared_ptr version     9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108; 9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108;
Success: inner product                     12; 12;
Success: outer product                     4 4 4; 4 4 4; 4 4 4; 4 4 4; 4 4 4; 4 4 4;
Success: + operator    0 4 9; 2 6 11; 4 8 13; 6 10 15; 0 4 9; 2 6 11; 4 8 13; 6 10 15;
Success: * operator (matrix multiplication) 9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108; 9 22 35 48; 11 30 49 68; 13 38 63 88; 15 46 77 108;
Success: Identity      1 0; 0 1; 1 0; 0 1;
Success: ninety        0 -1; 1 0; 0 -1; 1 0;
Success: ninety trasposed                  0 1; -1 0; 0 1; -1 0;
Success: thirty        0.866 -0.5; 0.5 0.866; 0.866 -0.5; 0.5 0.866;
Success: sixty         0.5 -0.866; 0.866 0.5; 0.5 -0.866; 0.866 0.5;
Success: thirty*sixty  0 -1; 1 0; 0 -1; 1 0;
Success: thirty*sixty* ninety (ninety is transposed)              1 0; 0 1; 1 0; 0 1;
Success: Rotation of vector using rotation matrix                 0.232; 3.6; 0.232; 3.6;

Rotation angle: 30degrees
0.866025 -0.5
0.5 0.866025

Rotation angle: -30degrees
0.866025 0.5
-0.5 0.866025
```