Cél:

- Konstruktorok: **standard, másoló (copy) és áthelyező (move)**
- Operátorok túlterhelése:
  - aritmetikai: **+, ***
  - inserter: **<<**
  - extractor: **>>**
  - index: **[ ]**
  - értékadás: **= (copy assignment, move assignment)**
- Standard kivételtípusok használata: **out_of_range** (**#include <stdexcept>**).

Adott a következő mátrix típus:

```
#ifndef MATRIX_H
#define MATRIX_H

#include <iostream>
#include <stdexcept>

using namespace std;

class Matrix {
public:
    //Methods
private:
    //Data
    double ** mElements;
    int mRows; //number of rows
    int mCols; //number of columns
};
#endif /* MATRIX_H */
```

A mátrix metódusai legyenek a következők (**ezeket helyezze az osztály publikus részébe!**):

Konstruktorok és destruktor:

```
    // Default constructor
    Matrix( int mRows = 10, int mCols =10);
    // Copy constructor
    Matrix(const Matrix& what);
    // Move constructor
    Matrix( Matrix&& what );
    // Destructor
    ~Matrix();
```

Feltöltő és kiíratő metódusok:

```
//fills the matrix with identical elements
void fillMatrix(double value);
//fills the matrix with random real numbers in the range [a, b)
void randomMatrix(int a, int b); //fills
//prints the matrix
void printMatrix(ostream& os = cout) const;
```

Lekérdező metódusok:

```
//checks whether this matrix is a square one
bool isSquare() const;
int getRows() const { return this->mRows;}
int getCols() const { return this->mCols;}
```

Operátorfüggvények (operátorok túlterhelése):

- **Összeadás és szorzás:**

```
// operation is permitted on matrices having the same dimensions
// otherwise throws an out_of_range exception!!
friend Matrix operator+(const Matrix& x, const Matrix& y);

// operation is permitted on matrices having proper dimensions
// otherwise throws an out_of_range exception!!
friend Matrix operator*(const Matrix& x, const Matrix& y);
```

- **Olvasás és írás:**

```
// extractor operator
friend istream & operator>>(istream& is, Matrix& mat);
// inserter operator
friend ostream & operator<<(ostream& os, const Matrix& mat);
```

- **Indexelés:**

```
// index operator
double* operator[] (int index);
// index operator that works on constant matrices!
double* operator[] (int index) const;
```

- **Értékadás operátorok:**

```
// Copy assignment
// operation is permitted between matrices having the same dimensions
// otherwise throws an exception (out_of_range)
Matrix & operator=(const Matrix& mat);
// Move assignment
Matrix & operator=(Matrix&& mat);
```

FEJLETT PROGRAMOZÁSI TECHNIKÁK (C++)
5. GYAKORLAT

Készítse el a **`Matrix.cpp`** osztályt, amelyben implementálja a fejállományban deklarált függvényeket. **Tesztelje az osztályt!**

Íme egy példa részleges tesztelésre:

```
#include <cstdlib>
#include "Matrix.h"

using namespace std;

Matrix createSquareMatrix(int size) {
    Matrix m(size, size);
    m.fillMatrix(1);
    return m;
}

int main(int argc, char** argv) {

    cout<<"**********************************************************************"<<endl;
    cout<<"Constructor "<<endl;
    cout<<"**********************************************************************"<<endl;

    Matrix m1(2, 3);
    m1.randomMatrix(1, 5);
    cout << "m1: " << endl << m1 << endl;


    cout<<"**********************************************************************"<<endl;
    cout<<"+ operator - equal sizes "<<endl;
    cout<<"**********************************************************************"<<endl;

    Matrix m2(2, 3);
    m2.fillMatrix(2);
    cout << "m2: " << endl << m2 << endl;
    try {
        cout << "Matrix m3 = m1 + m2: " << endl;
        Matrix m3 = (m1 + m2);
        cout << "m3: " << endl << m3 << endl;
    } catch (out_of_range& e) {
        cout << e.what() << endl;
    }

    cout<<"**********************************************************************"<<endl;
    cout<<"+ operator - different sizes "<<endl;
    cout<<"**********************************************************************"<<endl;

    Matrix m3(5, 5);
    m3.fillMatrix(1);
    cout << "m3: " << endl << m3 << endl;
    try {
        cout<<"m1+m3:"<<endl;
        cout << "m1+m3: " << endl << m1 + m3 << endl;
    } catch (out_of_range& e) {

        cout << e.what() << endl;
    }
```

```
cout<<"*****************************************************************"<<endl;
cout<<"copy assignment - different sizes "<<endl;
cout<<"*****************************************************************"<<endl;

try {
    //copy assignment
    cout<<"m3 = m1"<<endl;
    m3 = m1;
} catch (out_of_range& e) {
    cout << e.what() << endl;
}
cout<<"m3: "<<endl;
cout<<m3<<endl;


cout<<"*****************************************************************"<<endl;
cout<<"Extractor operator "<<endl;
cout<<"*****************************************************************"<<endl;

Matrix m4(1, 2);
cout << "Please type in two real numbers for m4[0][0] and m4[0][1]: " << endl;
//Extractor operator
cin>>m4;
//Inserter operator
cout << "m4: " << endl << m4 << endl;


cout<<"*****************************************************************"<<endl;
cout<<"Index operator "<<endl;
cout<<"*****************************************************************"<<endl;
//Index operator
cout << endl << "m4[0][0]: " << m4[0][0] << endl;

cout<<"*****************************************************************"<<endl;
cout<<"* operator "<<endl;
cout<<"*****************************************************************"<<endl;
Matrix m5(2, 1);
m5.fillMatrix(1);
cout << "M4: " << endl << m4 << endl;
cout << "M5: " << endl << m5 << endl;
cout << "Multiplication: ";
try {
    cout << "M4 x M5: " << endl << m4 * m5 << endl;
} catch (out_of_range& e) {
    cout << e.what() << endl;
}
cout<<"*****************************************************************"<<endl;
cout<<" = operator -- copy assignment "<<endl;
cout<<"*****************************************************************"<<endl;

Matrix m6(m4);
cout << "m6 created as a copy of m4 using copy constructor: " <<endl<< m6 << endl;
try {
    cout<<"m1 = m6 = m6"<<endl;
    m1 = m6 = m6;
} catch (out_of_range& e) {
    cout<< e.what() << endl;
}
```

```
    cout<<"*************************************************************"<<endl;
    cout<<"MOVE constructor "<<endl;
    cout<<"*************************************************************"<<endl;


    Matrix mx(3, 2), my(2, 3);
    mx.fillMatrix(1);
    my.fillMatrix(2);

    cout << "mx: " << endl << mx << endl;
    cout << "my: " << endl << my << endl;

    //Move constructor
    cout << "Matrix mz1 = std::move(mx * my);\n ";
    Matrix mz1 = std::move(mx * my);
    mz1.printMatrix(cout);


    Matrix mz2 = std::move(createSquareMatrix(3));
    cout << "Matrix mz2 = std::move(createSquareMatrix(3))\n ";
    mz2.printMatrix(cout);


    cout<<"*************************************************************"<<endl;
    cout<<"MOVE assignment "<<endl;
    cout<<"*************************************************************"<<endl;

    try {

        cout<<"mx: "<<mx.getRows()<<" x "<<mx.getCols()<<endl;
        cout<<mx<<endl;
        cout<<"my: "<<my.getRows()<<" x "<<my.getCols()<<endl;
        cout<<my<<endl;

        cout << "m6 = mx * my: " << endl;
        m6 = mx * my;
        cout<<"m6: "<<m6.getRows()<<" x "<<m6.getCols()<<endl;
        cout<<m6<<endl;
    } catch (out_of_range& e) {
        cout << e.what() << endl;
    }

    return 0;
}
```