

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

Name: Kadircan KURTULUŞ

Number: 16015001

Course: KOM3191 Object-Oriented Programming

Date: 6 December

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.

```
1  #include <iostream>
2  #include "Matrix.h"
3  using namespace std;
4  Matrix::Matrix()
5  {
6      rowN = 1;
7      colN = 1;
8      data = new float[1];
9      *data = 0;
10 }
11 Matrix::Matrix(const int rN, const int cN)
12 {
13     try
14     {
15         if (rN <= 0)
16             throw invalid_argument("Index in position 1 is invalid. Array
17                                     indices must be positive integers.");
18         else if (cN <= 0)
19             throw invalid_argument("Index in position 2 is invalid. Array
20                                     indices must be positive integers.");
21         else
22         {
23             rowN = rN;
24             colN = cN;
25             data = new float[rowN*colN];
26             for (int i = 0; i < rowN*colN; i++)
27                 data[i] = 0;
28         }
29     }
30     catch (const invalid_argument &ex)
31     {
32         cerr << ex.what() << endl;
33         rowN = 1;
34         colN = 1;
35         data = new float[1];
36         *data = 0;
37     }
38 }
39 Matrix::Matrix(const Matrix &srcMatrix) : Matrix(srcMatrix.rowN,
40                                                     srcMatrix.colN, srcMatrix.data) {}
41 Matrix::Matrix(const int rN, const int cN, const float const *srcPtr)
42 {
43     try
44     {
45         if (rN <= 0)
46             throw invalid_argument("Index in position 1 is invalid. Array
47                                     indices must be positive integers.");
48         else if (cN <= 0)
49             throw invalid_argument("Index in position 2 is invalid. Array
50                                     indices must be positive integers.");
51         else
52         {
53             rowN = rN;
54             colN = cN;
55             data = new float[rowN*colN];
56             for (int i = 0; i < rowN*colN; i++)
```

```
52         data[i] = srcPtr[i];
53     }
54 }
55 catch (const invalid_argument &ex)
56 {
57     cerr << ex.what() << endl;
58     rowN = 1;
59     colN = 1;
60     data = new float[1];
61     *data = 0;
62 }
63 }
64 const float* Matrix::getData()const
65 {
66     return data;
67 }
68 int Matrix::getRowN()const
69 {
70     return rowN;
71 }
72 int Matrix::getColN()const
73 {
74     return colN;
75 }
76 void Matrix::print()const
77 {
78     for (int i = 0; i < rowN; i++)
79     {
80         for (int j = 0; j < colN; j++)
81             cout << data[i*colN + j] << ' ';
82         cout << endl;
83     }
84 }
85 Matrix Matrix::transpose() const
86 {
87     Matrix temp(colN, rowN);
88     for (int i = 0; i < colN; i++)
89         for (int j = 0; j < rowN; j++)
90             temp.data[i*rowN + j] = data[i + j * colN];
91     return temp;
92 }
93 Matrix Matrix::operator+(const Matrix &rhsMatrix)const
94 {
95     Matrix temp(rowN, colN);
96     try
97     {
98         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
99             for (int i = 0; i < rowN*colN; i++)
100                 temp.data[i] = data[i] + rhsMatrix.data[i];
101         else
102             throw logic_error("Matrix dimensions must agree.");
103         return temp;
104     }
105     catch (const logic_error &ex)
106     {
107         cerr << ex.what() << endl;
```

```
108     return temp;
109 }
110 }
111 Matrix Matrix::operator-(const Matrix &rhsMatrix)const
112 {
113     Matrix temp(rowN, colN);
114     try
115     {
116         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
117             for (int i = 0; i < rowN*colN; i++)
118                 temp.data[i] = data[i] - rhsMatrix.data[i];
119         else
120             throw logic_error("Matrix dimensions must agree.");
121         return temp;
122     }
123     catch (const logic_error &ex)
124     {
125         cerr << ex.what() << endl;
126         return temp;
127     }
128 }
129 Matrix Matrix::operator*(const Matrix &rhsMatrix)const
130 {
131     Matrix temp(rowN, colN);
132     try
133     {
134         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
135             for (int i = 0; i < rowN*colN; i++)
136                 temp.data[i] = data[i] * rhsMatrix.data[i];
137         else
138             throw logic_error("Matrix dimensions must agree.");
139         return temp;
140     }
141     catch (const logic_error &ex)
142     {
143         cerr << ex.what() << endl;
144         return temp;
145     }
146 }
147 float Matrix::operator()(const int r, const int c)const
148 {
149     try
150     {
151         if (r <= 0)
152             throw invalid_argument("Index in position 1 is invalid. Array    ↗
153             indices must be positive integers.");
154         else if (r <= rowN)
155         {
156             if (c <= 0)
157                 throw invalid_argument("Index in position 2 is invalid. Array    ↗
158                 indices must be positive integers.");
159             else if (c <= colN)
160                 return data[(r - 1)*colN + c - 1];
161             else
162                 throw out_of_range("Index in position 2 exceeds array    ↗
163                 bounds.");
```

```
161     }
162     else
163         throw out_of_range("Index in position 1 exceeds array bounds.");
164 }
165 catch (const exception &ex)
166 {
167     cerr << ex.what() << endl;
168     return 0;
169 }
170 }
171 Matrix& Matrix::operator=(const Matrix &rhsMatrix)
172 {
173     rowN = rhsMatrix.rowN;
174     colN = rhsMatrix.colN;
175     data = new float[rhsMatrix.rowN*rhsMatrix.colN];
176     for (int i = 0; i < rhsMatrix.rowN*rhsMatrix.colN; i++)
177         data[i] = rhsMatrix.data[i];
178     return *this;
179 }
180 Matrix& Matrix::operator+=(const Matrix &rhsMatrix)
181 {
182     try
183     {
184         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
185             for (int i = 0; i < rowN*colN; i++)
186                 data[i] += rhsMatrix.data[i];
187         else
188             throw logic_error("Matrix dimensions must agree.");
189         return *this;
190     }
191     catch (const logic_error &ex)
192     {
193         cerr << ex.what() << endl;
194         return *this;
195     }
196 }
197 Matrix& Matrix::operator-=(const Matrix &rhsMatrix)
198 {
199     try
200     {
201         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
202             for (int i = 0; i < rowN*colN; i++)
203                 data[i] -= rhsMatrix.data[i];
204         else
205             throw logic_error("Matrix dimensions must agree.");
206         return *this;
207     }
208     catch (const logic_error &ex)
209     {
210         cerr << ex.what() << endl;
211         return *this;
212     }
213 }
214 Matrix& Matrix::operator*=(const Matrix &rhsMatrix)
215 {
216     try
```

```
217     {
218         if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
219             for (int i = 0; i < rowN*colN; i++)
220                 data[i] *= rhsMatrix.data[i];
221         else
222             throw logic_error("Matrix dimensions must agree.");
223         return *this;
224     }
225     catch (const logic_error &ex)
226     {
227         cerr << ex.what() << endl;
228         return *this;
229     }
230 }
231 int Matrix::operator==(const Matrix &rhsMatrix)const
232 {
233     if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
234     {
235         for (int i = 0; i < rowN*colN; i++)
236             if (data[i] != rhsMatrix.data[i])
237                 return 0;
238         return 1;
239     }
240     return 0;
241 }
242 int Matrix::operator!=(const Matrix &rhsMatrix)const
243 {
244     if (rowN == rhsMatrix.rowN && colN == rhsMatrix.colN)
245     {
246         for (int i = 0; i < rowN*colN; i++)
247             if (data[i] != rhsMatrix.data[i])
248                 return 1;
249         return 0;
250     }
251     return 1;
252 }
```