

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.

```
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     rowN = rN;
14     colN = cN;
15     data = new float[rowN*colN];
16     for (int i = 0; i < rowN; i++)
17         for (int j = 0; j < colN; j++)
18             data[i*colN + j] = 0;
19 }
20 Matrix::Matrix(const Matrix &srcMatrix)
21 {
22     rowN = srcMatrix.rowN;
23     colN = srcMatrix.colN;
24     data = srcMatrix.data;
25 }
26 Matrix::Matrix(const int rN, const int cN, const float const *srcPtr)
27 {
28     rowN = rN;
29     colN = cN;
30     data = new float[rowN*colN];
31     for (int i = 0; i < rowN; i++)
32         for (int j = 0; j < colN; j++)
33             data[i*colN + j] = *(srcPtr++);
34 }
35 const float* Matrix::getData()const
36 {
37     return data;
38 }
39 int Matrix::getRowN()const
40 {
41     return rowN;
42 }
43 int Matrix::getColN()const
44 {
45     return colN;
46 }
47 void Matrix::print()const
48 {
49     for (int i = 0; i < rowN; i++)
50     {
51         for (int j = 0; j < colN; j++)
52             cout << data[i*colN + j] << ' ';
53         cout << endl;
54     }
55 }
56 Matrix Matrix::transpose() const
```

```
57 {
58     Matrix temp(colN, rowN);
59     for (int i = 0; i < colN; i++)
60         for (int j = 0; j < rowN; j++)
61             temp.data[i*rowN + j] = data[i + j*colN];
62     return temp;
63 }
64 Matrix Matrix::operator+(const Matrix &rhsMatrix)const
65 {
66     try
67     {
68         if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
69         {
70             Matrix temp(rowN, colN);
71             for (int i = 0; i < rowN; i++)
72                 for (int j = 0; j < colN; j++)
73                     temp.data[i*colN + j] = data[i*colN + j] + rhsMatrix.data
74                                     [i*colN + j];
75             return temp;
76         }
77         else
78             throw logic_error("Matrix dimensions must agree.");
79     }
80     catch (const logic_error &ex)
81     {
82         cerr << ex.what() << endl;
83     }
84 }
85 Matrix Matrix::operator-(const Matrix &rhsMatrix)const
86 {
87     try
88     {
89         if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
90         {
91             Matrix temp(rowN, colN);
92             for (int i = 0; i < rowN; i++)
93                 for (int j = 0; j < colN; j++)
94                     temp.data[i*colN + j] = data[i*colN + j] - rhsMatrix.data
95                                     [i*colN + j];
96             return temp;
97         }
98         else
99             throw logic_error("Matrix dimensions must agree.");
100     }
101     catch (const logic_error &ex)
102     {
103         cerr << ex.what() << endl;
104     }
105 }
106 Matrix Matrix::operator*(const Matrix &rhsMatrix)const
107 {
108     try
109     {
110         if (colN == rhsMatrix.colN && rowN == rhsMatrix.rowN)
111         {
112             Matrix temp(rowN, colN);
```

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

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

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