Software Engineering
Lab 8 Report

Emre Ozan Alkan
{emreozanalkan@gmail.com}
MSCV-5

7 December 2013

1 Implementing CMatrix

In this lab we implemented CMatrix class to simulate a matrix and its operations like '+', '-', '*', etc.

1.1 CMatrix Layout

Here is the how CMatrix class looks like:

Listing 1: CMatrix Overview

```
1  template<class T = int>
2  class CMatrix
3  {
4  private:
5      vector<vector<T> > *matrix;
6      int m;
7      int n;
8  public:
9    CMatrix();
10   CMatrix(int size);
11   CMatrix(int m, int n);
12   ~CMatrix();
13
14   void randomize();
15   void zeros();
16   void identity();
17
18   inline vector<vector<T> > getMatrix() const;
19   inline void setMatrix(vector<vector<T> > *matrix);
20   inline int getM() const;
21   inline void setM(int m);
22   inline int getN() const;
23   inline void setN(int n);
24
25
26   void fill();
27   friend istream& operator>>(istream& istream, CMatrix* matrix);
28   friend istream& operator>>(istream& istream, CMatrix& matrix);
29
30   void display() const;
31   friend ostream& operator<<(ostream& ostream, const CMatrix* matrix);
32   friend ostream& operator<<(ostream& ostream, const CMatrix& matrix);
33
```

```
34    void transpose();
35
36    T trace() const;
37
38    bool operator==(const CMatrix<T>& cmatrix) const;
39
40    CMatrix<T> operator+(const CMatrix<T>& cmatrix) const;
41    CMatrix<T> operator-(const CMatrix<T>& cmatrix) const;
42    CMatrix<T> operator*(const CMatrix<T>& cmatrix) const;
43 };
```

1.2 Constructors

We implemented 3 constructors. Default constructor, constructor with one parameter building identity matrix and constructor getting size of matrix and initializing it with random values.

Listing 2: CMatrix Constructors

```
1
2     // Default Constructor: Inits empty matrix 0x0 sizes.
3     CMatrix() : m(0), n(0)
4     {
5         matrix = new vector<vector<T> >(m, vector<T>(n, 0));
6     }
7
8     // Constructor: Getting size to build square matrix of size x size.
9     // Then matrix set to identity
10    CMatrix(int size) : m(size), n(size)
11    {
12        matrix = new vector<vector<T> >(m, vector<T>(n, 0));
13        identity(); // sets matrix to identity
14    }
15
16    // Constructor: Inits the matrix with the given sizes m and n
17    // Then randomly assigning values to matrix.
18    CMatrix(int m, int n) : m(m), n(n)
19    {
20        matrix = new vector<vector<T> >(m, vector<T>(n));
21        randomize(); // fills the matrix with random values
22    }
23
24    // Destructor: Cleaning our variables created with new keyword.
25    ~CMatrix()
26    {
27        clog<<"CMatrix deleting..."<<endl;
28        delete matrix;
29        matrix = 0;
30    }
```

1.3 User Input and >> overload

We created fill function to get input from user to enter coefficients of the matrix. Also this is called when >> operator is used on CMatrix.

Listing 3: CMatrix Input

```
1
2     // Fills the matrix with the user input
3     void fill()
4     {
5         int i = 0, j;
```

```
6           for(typename vector<vector<T> >::iterator row_it = matrix->begin(); row_it !=
                matrix->end(); row_it++, i++)
7           {
8               j = 0;
9               for(typename vector<T>::iterator col_it = (*row_it).begin(); col_it != (*
                    row_it).end(); col_it++, j++)
10              {
11                  cout<<"Please enter matrix["<<i<<"]["<<j<<"]: ";
12                  cin>>(*col_it);
13              }
14          }
15      }
16
17      // operator overloading for ">>"
18      friend istream& operator>>(istream& istream, CMatrix* matrix)
19      {
20          matrix->fill();
21          return istream;
22      }
23
24      // operator overloading for ">>"
25      friend istream& operator>>(istream& istream, CMatrix& matrix)
26      {
27          matrix.fill();
28          return istream;
29      }
```

1.4 Display and << overload

CMatrix also overloaded for << operator to display matrix itself with display function.

Listing 4: CMatrix Display

```
1
2       // Displays the current matrix in matrix form.
3       void display() const
4       {
5           for(typename vector<vector<T> >::iterator row_it = matrix->begin(); row_it !=
                matrix->end(); row_it++)
6           {
7               for(typename vector<T>::iterator col_it = (*row_it).begin(); col_it != (*
                    row_it).end(); col_it++)
8                   cout<<*col_it<<" ";
9
10              cout<<endl;
11          }
12      }
13
14      // operator overloading for "<<"
15      friend ostream& operator<<(ostream& ostream, const CMatrix* matrix)
16      {
17          matrix->display();
18          return ostream;
19      }
20
21      // operator overloading for "<<"
22      friend ostream& operator<<(ostream& ostream, const CMatrix& matrix)
23      {
24          matrix.display();
25          return ostream;
26      }
```

1.5 Transpose

We created transpose function as requested. In this function I check the current size, if it is square matrix, it getting transpose as in-place, otherwise creating another temporary matrix to hold transpose and changing it with current one.

Listing 5: CMatrix Transpose

```
1
2       // Transposes the matrix. If square matrix,
3       // just changing values across diagonal, otherwise
4       // initializing new matrix with N x M, and put traspose
5       // into it.
6       void transpose()
7       {
8           // if square matrix, no need to create another matrix
9           // change operations on same matrix, std::swap is used
10          if(m == n)
11              for(int i = 0; i < m; i++)
12                  for(int j = 0; j < i; j++)
13                      std::swap((*matrix)[i][j], (*matrix)[j][i]);
14          else
15          {
16              // if matrix is not square, we need to craete another matrix M x N to N x M
17              vector<vector<T> > *newMatrix = new vector<vector<T> >(n, vector<T>(m, 0));
18
19              for(int i = 0; i < m; i++)
20                  for(int j = 0; j < n; j++)
21                      (*newMatrix)[j][i] = (*matrix)[i][j];
22
23              delete matrix;
24              matrix = newMatrix;
25              newMatrix = 0;
26              std::swap(m, n);
27          }
28      }
```

1.6 Overloads for +, - and *

As all applications with matrices support addition, subtraction and multiplication, we overloaded these operators to provide similar functionality.

Listing 6: CMatrix Basic Operations

```
1
2       // member operator overloading for "+"
3       CMatrix<T> operator+(const CMatrix<T>& cmatrix) const
4       {
5           if(this->m != cmatrix.m || this->n != cmatrix.n)
6           {
7               cerr<<"ERROR: Matrix sizes are not equal for addition..."<<endl<<"Terminating
                    ..."<<endl;
8               //return 0;
9               exit(EXIT_FAILURE);
10          }
11
12          CMatrix<T> tempMatrix(m, n);
13
14          typename vector<vector<T> >::iterator row_it_current;
15          typename vector<vector<T> >::iterator row_it_cmatrix;
16          typename vector<vector<T> >::iterator row_it_tempmatrix;
17
18          for(row_it_current = matrix->begin(),
19              row_it_cmatrix = cmatrix.matrix->begin(),
```

```
20              row_it_tempmatrix = tempMatrix.matrix->begin();
21
22              row_it_current != matrix->end() &&
23              row_it_cmatrix != cmatrix.matrix->end() &&
24              row_it_tempmatrix != tempMatrix.matrix->end();
25
26              row_it_current++,
27              row_it_cmatrix++,
28              row_it_tempmatrix++)
29          {
30              typename vector<T>::iterator col_it_current;
31              typename vector<T>::iterator col_it_cmatrix;
32              typename vector<T>::iterator col_it_tempmatrix;
33
34              for(col_it_current = (*row_it_current).begin(),
35                  col_it_cmatrix = (*row_it_cmatrix).begin(),
36                  col_it_tempmatrix = (*row_it_tempmatrix).begin();
37
38                  col_it_current != (*row_it_current).end() &&
39                  col_it_cmatrix != (*row_it_cmatrix).end() &&
40                  col_it_tempmatrix != (*row_it_tempmatrix).end();
41
42                  col_it_current++,
43                  col_it_cmatrix++,
44                  col_it_tempmatrix++)
45              {
46                  *col_it_tempmatrix = *col_it_current + *col_it_cmatrix;
47              }
48
49          }
50
51          return tempMatrix;
52      }
53
54      // member operator overloading for "-"
55      CMatrix<T> operator-(const CMatrix<T>& cmatrix) const
56      {
57          if(this->m != cmatrix.m || this->n != cmatrix.n)
58          {
59              cerr<<"ERROR: Matrix sizes are not equal for substraction..."<<endl<<"
                    Terminating..."<<endl;
60              //return 0;
61              exit(EXIT_FAILURE);
62          }
63
64          CMatrix<T> tempMatrix(m, n);
65
66          typename vector<vector<T> >::iterator row_it_current;
67          typename vector<vector<T> >::iterator row_it_cmatrix;
68          typename vector<vector<T> >::iterator row_it_tempmatrix;
69
70          for(row_it_current = matrix->begin(),
71              row_it_cmatrix = cmatrix.matrix->begin(),
72              row_it_tempmatrix = tempMatrix.matrix->begin();
73
74              row_it_current != matrix->end() &&
75              row_it_cmatrix != cmatrix.matrix->end() &&
76              row_it_tempmatrix != tempMatrix.matrix->end();
77
78              row_it_current++,
79              row_it_cmatrix++,
80              row_it_tempmatrix++)
81          {
82              typename vector<T>::iterator col_it_current;
83              typename vector<T>::iterator col_it_cmatrix;
84              typename vector<T>::iterator col_it_tempmatrix;
85
86              for(col_it_current = (*row_it_current).begin(),
87                  col_it_cmatrix = (*row_it_cmatrix).begin(),
```

```
88                  col_it_tempmatrix = (*row_it_tempmatrix).begin();
89
90                  col_it_current != (*row_it_current).end() &&
91                  col_it_cmatrix != (*row_it_cmatrix).end() &&
92                  col_it_tempmatrix != (*row_it_tempmatrix).end();
93
94                  col_it_current++,
95                  col_it_cmatrix++,
96                  col_it_tempmatrix++)
97              {
98                  *col_it_tempmatrix = *col_it_current - *col_it_cmatrix;
99              }
100
101         }
102
103         return tempMatrix;
104     }
105
106     // member operator overloading for "*"
107     CMatrix<T> operator*(const CMatrix<T>& cmatrix) const
108     {
109         if(this->n != cmatrix.m)
110         {
111             cerr<<"ERROR: Matrix sizes are not equal for multiplication..."<<endl<<"
                    Terminating..."<<endl;
112             //return 0;
113             exit(EXIT_FAILURE);
114         }
115
116         CMatrix<T> tempMatrix(this->m, cmatrix.n);
117         tempMatrix.zeros();
118
119         for(int i = 0; i < this->m; i++){
120             for(int j = 0; j < cmatrix.n; j++){
121                 for(int k = 0; k < cmatrix.m; k++){
122                     (*tempMatrix.matrix)[i][j] += (*matrix)[i][k] * (*cmatrix.matrix)[k][
                            j];
123                 }
124             }
125         }
126
127         return tempMatrix;
128     }
```

1.7 Overload for ==

We overloaded "==" operator to check if two matrices are same. If sizes are different, it returns immediately with result otherwise, it compare two matrices element wise.

Listing 7: CMatrix Equal Check

```
1
2     // member operator overloading for "=="
3     bool operator==(const CMatrix<T>& cmatrix) const
4     {
5         if(this->m != cmatrix.m || this->n != cmatrix.n)
6             return false;
7
8         bool isEqual = true;
9
10        typename vector<vector<T> >::iterator row_it_current;
11        typename vector<vector<T> >::iterator row_it_cmatrix;
12
13        for(row_it_current = matrix->begin(), row_it_cmatrix = cmatrix.matrix->begin();
              row_it_current != matrix->end() && row_it_cmatrix != cmatrix.matrix->end();
              row_it_current++, row_it_cmatrix++)
```

```
14          {
15  //              typename vector<T>::iterator col_it_current;
16  //              typename vector<T>::iterator col_it_cmatrix;
17
18  //              for(col_it_current = (*row_it_current).begin(), col_it_cmatrix = (*
        row_it_cmatrix).begin(); col_it_current != (*row_it_current).end() && col_it_cmatrix
        != (*row_it_cmatrix).end(); col_it_current++, col_it_cmatrix++)
19  //                  if(*col_it_current != *col_it_cmatrix)
20  //                      return false;
21
22              isEqual = std::equal((*row_it_current).begin(), (*row_it_current).end(), (*
                    row_it_cmatrix).begin());
23              if(!isEqual)
24                  break;
25          }
26
27          return isEqual;
28      }
```

1.8 Trace

We also asked to implement CMatrix function for calculating trace of the matrix. It consecutively adding up diagonal elements of the matrix.

Listing 8: CMatrix Trace

```
1
2       // member operator overloading for "*"
3       CMatrix<T> operator*(const CMatrix<T>& cmatrix) const
4       {
5           if(this->n != cmatrix.m)
6           {
7               cerr<<"ERROR: Matrix sizes are not equal for multiplication..."<<endl<<"
                    Terminating..."<<endl;
8               //return 0;
9               exit(EXIT_FAILURE);
10          }
11
12          CMatrix<T> tempMatrix(this->m, cmatrix.n);
13          tempMatrix.zeros();
14
15          for(int i = 0; i < this->m; i++){
16              for(int j = 0; j < cmatrix.n; j++){
17                  for(int k = 0; k < cmatrix.m; k++){
18                      (*tempMatrix.matrix)[i][j] += (*matrix)[i][k] * (*cmatrix.matrix)[k][
                            j];
19                  }
20              }
21          }
22
23          return tempMatrix;
24      }
```

1.9 Helper Functions

Here is the some other function implemented in CMatrix as helper functions.

Listing 9: CMatrix Helper Functions

```
1
2       // Randomly initialize the matrix that inside the class between 0 and 5
```

```
3       void randomize()
4       {
5           for(typename vector<vector<T> >::iterator row_it = matrix->begin(); row_it !=
                matrix->end(); row_it++)
6               for(typename vector<T>::iterator col_it = (*row_it).begin(); col_it != (*
                    row_it).end(); col_it++)
7                   *col_it = T(rand() % 5);
8       }
9
10      // Fill the matrix with zeros.
11      void zeros()
12      {
13          for(typename vector<vector<T> >::iterator row_it = matrix->begin(); row_it !=
                matrix->end(); row_it++)
14              for(typename vector<T>::iterator col_it = (*row_it).begin(); col_it != (*
                    row_it).end(); col_it++)
15                  *col_it = T(0);
16      }
17
18      // Converts current matrix to identity matrix
19      void identity()
20      {
21          if(m != n)
22              clog<<"identity() on non-square matrix"<<endl;
23
24          zeros();
25
26          // We need to know minimum of the size to traverse on diagonal
27          int diogonalCount = std::min(m, n);
28
29          typename vector<vector<T> >::iterator it_row;
30          typename vector<T>::iterator it_col;
31
32          // We going to iterate diogonalCount times our iterators
33          for(int i = 0; i < diogonalCount; i++)
34          {
35              it_row = matrix->begin();
36              for(int j = 0; j < i; j++) // iterate through current diagonal element for
                    row
37                  ++it_row;
38
39              it_col = (*it_row).begin();
40              for(int k = 0; k < i; k++) // iterate through current diagonal element for
                    column
41                  ++it_col;
42
43              *it_col = T(1); // set the (i, i). diogonal element to 1
44          }
45      }
```

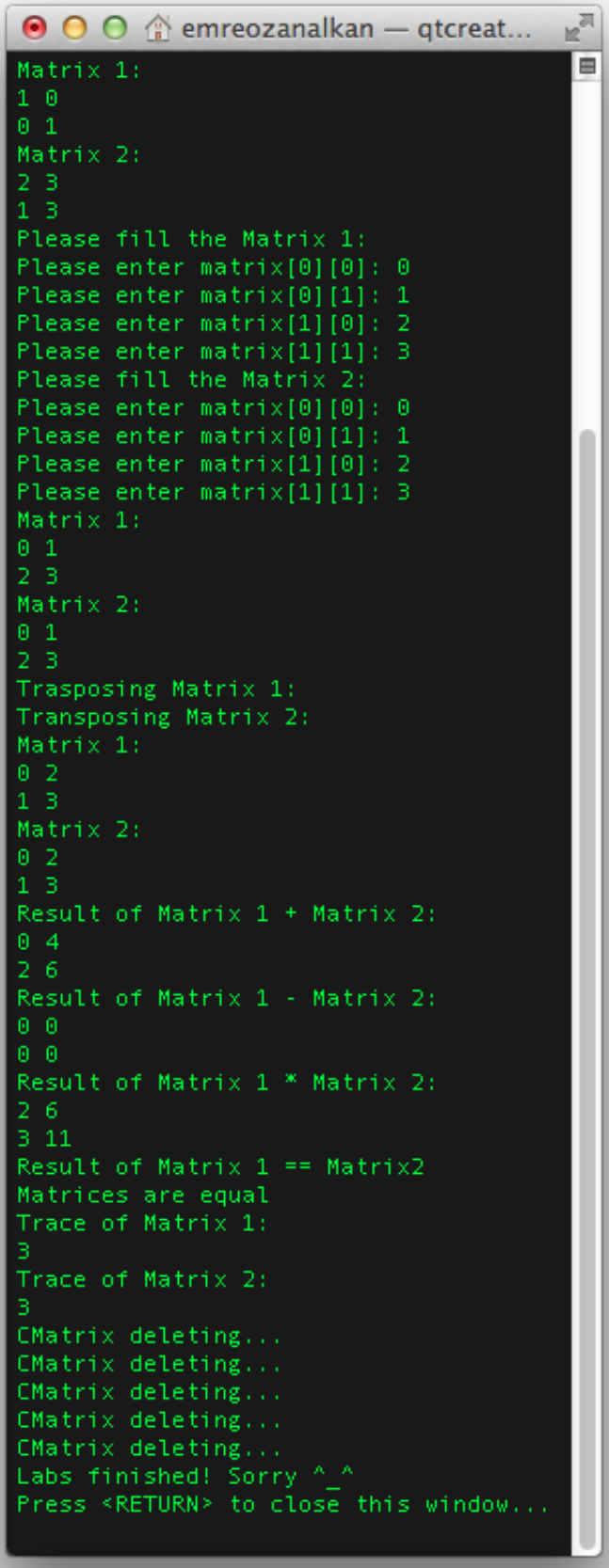
1.10 Result

Here is the example main and the output of the program.

Listing 10: Example Main

```
1
2   int main(int argc, char **argv, char **env)
3   {
4       atexit(allLabsFinished);
5       path = argv[0];
6
7       srand(time(NULL));
8
9       CMatrix<> matrix1(2);
10      CMatrix<> matrix2(2, 2);
```

```
11
12      cout<<"Matrix 1:"<<endl;
13      cout<<matrix1;
14      cout<<"Matrix 2:"<<endl;
15      matrix2.display();
16
17      cout<<"Please fill the Matrix 1:"<<endl;
18      matrix1.fill();
19      cout<<"Please fill the Matrix 2:"<<endl;
20      cin>>matrix2;
21
22      cout<<"Matrix 1:"<<endl;
23      matrix1.display();
24      cout<<"Matrix 2:"<<endl;
25      cout<<matrix2;
26
27      cout<<"Trasposing Matrix 1:"<<endl;
28      matrix1.transpose();
29      cout<<"Transposing Matrix 2:"<<endl;
30      matrix2.transpose();
31
32      cout<<"Matrix 1:"<<endl;
33      cout<<matrix1;
34      cout<<"Matrix 2:"<<endl;
35      matrix2.display();
36
37      auto matrixAdd = matrix1 + matrix2;
38      auto matrixSub = matrix1 - matrix2;
39      auto matrixMul = matrix1 * matrix2;
40      auto matrixEql = (matrix1 == matrix2);
41      auto matrix1Trace = matrix1.trace();
42      auto matrix2Trace = matrix2.trace();
43
44      cout<<"Result of Matrix 1 + Matrix 2:"<<endl;
45      matrixAdd.display();
46      cout<<"Result of Matrix 1 - Matrix 2:"<<endl;
47      cout<<matrixSub;
48      cout<<"Result of Matrix 1 * Matrix 2:"<<endl;
49      matrixMul.display();
50      cout<<"Result of Matrix 1 == Matrix2"<<endl;
51      if(matrixEql)
52          cout<<"Matrices are equal"<<endl;
53      else
54          cout<<"Matrices are NOT equal"<<endl;
55      cout<<"Trace of Matrix 1:"<<endl;
56      cout<<matrix1Trace<<endl;
57      cout<<"Trace of Matrix 2:"<<endl;
58      cout<<matrix2Trace<<endl;
59
60      return 0;
61  }
```


emreozanalkan — qtcreat...
Matrix 1:
1 0
0 1
Matrix 2:
2 3
1 3
Please fill the Matrix 1:
Please enter matrix[0][0]: 0
Please enter matrix[0][1]: 1
Please enter matrix[1][0]: 2
Please enter matrix[1][1]: 3
Please fill the Matrix 2:
Please enter matrix[0][0]: 0
Please enter matrix[0][1]: 1
Please enter matrix[1][0]: 2
Please enter matrix[1][1]: 3
Matrix 1:
0 1
2 3
Matrix 2:
0 1
2 3
Trasposing Matrix 1:
Transposing Matrix 2:
Matrix 1:
0 2
1 3
Matrix 2:
0 2
1 3
Result of Matrix 1 + Matrix 2:
0 4
2 6
Result of Matrix 1 - Matrix 2:
0 0
0 0
Result of Matrix 1 * Matrix 2:
2 6
3 11
Result of Matrix 1 == Matrix2
Matrices are equal
Trace of Matrix 1:
3
Trace of Matrix 2:
3
CMatrix deleting...
CMatrix deleting...
CMatrix deleting...
CMatrix deleting...
CMatrix deleting...
Labs finished! Sorry ^_^
Press <RETURN> to close this window...