Şafak AKINCI
152120141003

27.10.2016

Experiment 2
Introduction to C++ Programming – II

Objectives

To write a simple computer programs like creating a matrix library in C++.

Prelab Activities

Lab Exercise 1 – Matrix Library Creation

```
/****************************************
 * Matrix.h                             *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 2: Introduction to C++ - II*
 ****************************************/

//  Created Matrix.h (header file) and it includes given function prototypes.

#pragma once
/*
 In the C and C++ programming languages, " #pragma once " is a non-standard
    but widely supported preprocessor directive designed to cause the current source file
    to be included only once in a single compilation.
 " #pragma once " has several advantages, including:
    less code, avoidance of name clashes, and sometimes improvement in compilation speed.

*** In this case, because of the #pragma once preprocessor directive (it works for Visual Studio),
*** Matrix.h file will compile just for once.
*/

//  Matrix (the type of struct) is created in STACK, and it has three members
//  Two of them are "int" and the other one is "pointer to point another pointer". (**)
struct Matrix{
    int rowSize = -1;
    int columnSize = -1;
    float** data = 0;
};

//  Allocating memory to matrix for required sizes.
void Matrix_Allocate(Matrix& matrix, int rowSize, int columnSize);

//  Deleting the elements of the given matrix.
void Matrix_Free(Matrix& matrix);

//  Filling the given matrix with the given value.
void Matrix_FillByValue(Matrix& matrix, float value);

//  Assigning data's elements to matrix.
void Matrix_FillByData(Matrix& matrix, float** data);

//  Displaying all elements of the given matrix.
void Matrix_Display(const Matrix& matrix);

//  Adding two matrices and saving the result to result matrix.
void Matrix_Addition(const Matrix& matrix_left, const Matrix& matrix_right, Matrix& result);

//  Subtracting two matrices and saving the result to result matrix.
void Matrix_Substruction(const Matrix& matrix_left, const Matrix& matrix_right, Matrix& result);
```

```
//  Multiplication the given matrices and saving the result to result matrix.
void Matrix_Multiplication(const Matrix& matrix_left, const Matrix& matrix_right, Matrix& result);

//  Multiplication the given matrix with scalarValue and saving the result to result matrix.
void Matrix_Multiplication(const Matrix& matrix_left, float scalarValue, Matrix& result);

//  Dividing the given matrix to scalarValue and saving the result to result matrix.
void Matrix_Division(const Matrix& matrix_left, float scalarValue, Matrix& result);

//  Transposing the given matrix and saving it to result matrix.
void Matrix_Transpose(const Matrix& matrix, Matrix&result);

//  Calculating the row module of the given matrix and saving it to result matrix.
void Matrix_Row_Module(const Matrix& matrix, Matrix&result);

//  Calculating the column module of the given matrix and saving it to result matrix.
void Matrix_Column_Module(const Matrix& matrix, Matrix&result);
```

```
/*****************************************
 * Matrix.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 2: Introduction to C++ - II*
 * Lab Exercise: Matrix Library Creation *
 *****************************************/

//  Created Matrix.cpp and included the Matrix.h .
//  A C++ Matrix library that performs the basic matrix operations
//  by using the given struct type (Matrix) is written and tested.

//  Implemented required functions that located in the Matrix.h file.

#include "Matrix.h"                 //Adding Matrix.h header file that include function prototypes.
#include <iostream>                 //Adding iostream header to use standart input output functions.
#include <iomanip>                  //Adding iomanip header to use setw() function but didn't use.
#include <math.h>                   //Adding math.h header to use pow and sqrt functions.

using namespace std;                //To don't write for each code std::   (e.x. std::cout)

//  Matrix_Allocate function takes a reference called matrix the type of the struct Matrix
//  which is defined in Matrix.h and two variables called rowSize and columnSize to allocate
//  memory to dynamic matrix.
//  All parameters were taken from MatrixTestApp.cpp .
/*  Function took reference to don't waste memory (like copying given parameter for function). */
/*  Using reference allows us to change(assign etc) given parameter. */
/*  We can access the given parameter under the name of "matrix" in Matrix_Allocate function */

/***  This function will allocate two-dimensional dynamic array into the data member of the Matrix
    and update rowSize and columnSize variables which are the member of matrix. ***/

void Matrix_Allocate(Matrix& matrix, int rowSize, int columnSize)
{

    //rowSize(2) and columnSize(3) are assigned to matrix's members called rowSize and columnSize.
    matrix.rowSize = rowSize;
    matrix.columnSize = columnSize;

    //A dynamic two-dimensional array is basically "an array of pointers to pointers".
```

```
    //  matrix.data is a pointer which is created in STACK will point another pointer.

    //  rowSize times elements that all of them are float* (float pointer) are created in HEAP MEMORY,
    //  and assigned to matrix's member called data (float**).

    matrix.data = new float* [matrix.rowSize];

    //To get two-dimensional array, each ROW of the array must have columnSize times elements(float).
    for(int i=0; i<matrix.rowSize; i++)
        matrix.data[i] = new float [matrix.columnSize];

}//end Matrix_Allocate ()

//  Matrix_Free function takes a reference called matrix the type of the struct Matrix which is defined in Matrix.h
//  Matrix_Free function will delete the elements of the given parameter, in this case it is called matrix.
/***  This function will free the allocated memory for the given Matrix to prevent memory leak
    and assign rowSize and columnSize which are the member of matrix to -1 and data to nullptr. ***/
void Matrix_Free(Matrix& matrix)
{

    //  Deleted all columns of matrix for each row.
    for(int i=0; i<matrix.rowSize; i++){
        delete[]matrix.data[i];
    }

    //  Deleted all rows of matrix.
    delete[]matrix.data;

    //  -1 is assigned to matrix.rowSize and matrix.columnSize to know matrix.data isn't initialized (empty).
    matrix.rowSize = -1;
    matrix.columnSize = -1;
    //  matrix.data is a pointer, after the removing its items, it should point to NULL.
    matrix.data = nullptr;

}//end Matrix_Free ()

//  Matrix_FillByValue takes a reference called matrix the type of the stuct Matrix which is defined in Matrix.h
//  and a float variable called value which is equal to 1.34 was given by MatrixTestApp.cpp .
/***  This function will fill the data member of the Matrix by the given value. ***/
void Matrix_FillByValue(Matrix& matrix, float value)
{
    //  value(1.34) is assigned to all elements of the matrix.
    for(int i=0; i<matrix.rowSize; i++){
        for(int j=0; j<matrix.columnSize; j++){
            matrix.data[i][j]=value;
        }//end for
    }//end FOR

}//end Matrix_FillByValue ()

//  Matrix_FillByData takes one reference called matrix, and one 2-D array called data
//  which is generated from GetRandomData() function that located in MatrixTestApp.cpp .
/***  This function will fill the data member of the Matrix by
    the corresponding elements of the given two-dimensional array.  ***/
void Matrix_FillByData(Matrix & matrix, float ** data)
{

    //  data's elements are assigned to matrix.data .
    for(int i=0; i<matrix.rowSize; i++){
        for(int j=0; j<matrix.columnSize; j++){
            matrix.data[i][j]=data[i][j];
        }//end for
```

```
    }//end FOR

}//end Matrix_FillByData

//  Matrix_FillByValue takes "a constant(to don't change anything of its) reference" called matrix
//"const" word is used to prevent changing the data of the given matrix, just get(read) its data.
//  the type of the stuct Matrix which is defined in Matrix.h.
/***  This function will display the matrix's elements to console. ***/

void Matrix_Display(const Matrix& matrix)
{

    cout<<"\nMATRIX:\t"<<matrix.rowSize<<" x "<<matrix.columnSize<<endl<<endl;

    //  All elements of the array will print to console.
    for(int i=0; i<matrix.rowSize; i++){
        for(int j=0; j<matrix.columnSize; j++){
            cout<<"\t"<<matrix.data[i][j]<<"\t\t";
        }//end for
        cout<<endl;
    }//end FOR

}//end Matrix_Display ()

//  Matrix_Addition takes three references that two of them are constant are the type of struct called Matrix.
//"const" word is used to prevent changing the data of the given matrix, just get(read) its data.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
and perform matrix addition for the given first two matrices by saving the result into the Matrix named result.*/
void Matrix_Addition(const Matrix & matrix_left, const Matrix & matrix_right, Matrix &result)
{

    //  Matrix_Allocate function allocate memory for result.
    Matrix_Allocate(result, matrix_left.rowSize, matrix_left.columnSize);

    //  The addition of the two matrices will calculate and assign to result (Matrix).
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = matrix_left.data[i][j] + matrix_right.data[i][j];
        }//end for
    }//end FOR

}//end Matrix_Addition ()

//  Matrix_Substruction takes three references that two of them are constant are the type of struct called Matrix.
//"const" word is used to prevent changing the data of the given matrix, just get(read) its data.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
and perform matrix subtruction for the given first two matrices by saving the result into the Matrix named result.*/
void Matrix_Substruction(const Matrix & matrix_left, const Matrix & matrix_right, Matrix & result)
{

    //  Matrix_Allocate function allocate memory for result.
    Matrix_Allocate(result, matrix_left.rowSize, matrix_left.columnSize);

    //  The "subtraction" of the two matrices will calculate and assign to result (Matrix).
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = matrix_left.data[i][j] - matrix_right.data[i][j];
        }//end for
    }//end FOR

}//end Matrix_Substruction ()

//  Matrix_Multiplication takes three references that two of them are constant are the type of struct called Matrix.
```

```
//"const" word is used for don't change the data of the given matrix, just get(read) its data.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
and perform matrix multiplication for the given first two matrices by saving the result into the Matrix named result*/
void Matrix_Multiplication(const Matrix & matrix_left, const Matrix & matrix_right, Matrix & result)
{

    //  Matrix_Allocate function allocate memory for result.
    Matrix_Allocate(result, matrix_left.rowSize, matrix_right.columnSize);

    //  The multiplication of the two matrices will calculate and assign to result (Matrix).
    for(int i=0;i<matrix_left.rowSize;i++){
        for(int j=0;j<matrix_right.columnSize;j++){

            result.data[i][j]=0;

            for(int k=0;k<matrix_left.columnSize;k++){
                result.data[i][j] += matrix_left.data[i][k] * matrix_right.data[k][j];
            }//end for

        }//end FOR
    }//END FOR

}//end Matrix_Multiplication ()

//  Matrix_Multiplication takes two references that their types are struct called Matrix
//and one float variable called scalarValue.
//  There are two function with same name (Matrix_Multiplication), the difference between them
//are variables that they take.
//  Program will decide which function should call according to their variables
//when the Matrix_Multiplication function call.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
 and perform multiplication with matrix and the scalarValue(float) by saving the result into the Matrix named result*/
void Matrix_Multiplication(const Matrix & matrix_left, float scalarValue, Matrix & result)
{

    //  Matrix_Allocate function allocates memory for result.
    Matrix_Allocate(result, matrix_left.rowSize, matrix_left.columnSize);

    //  matrix_left.data will multiple with the scalarValue (float) and assign to result.data .
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = matrix_left.data[i][j] * scalarValue;
        }//end for
    }//end FOR

}//end Matrix_Multiplication ()

//  Matrix_Division takes two references that their types are struct called Matrix
//and one float variable called scalarValue.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
 and perform divison with matrix and the scalarValue(float) by saving the result into the Matrix named result*/
void Matrix_Division(const Matrix & matrix_left, float scalarValue, Matrix & result)
{

    //  Matrix_Allocate function allocates memory for result.
    Matrix_Allocate(result, matrix_left.rowSize, matrix_left.columnSize);

    //  matrix_left.data will divide to the scalarValue (float) and assign to result.data .
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = matrix_left.data[i][j] / scalarValue;
        }//end for
    }//end FOR
```

```
}//end Matrix_Division ()

//  Matrix_Transpose takes two references(one is const) that their types are struct called Matrix.
/***  This function will call Matrix_Allocate function to create result array for the required sizes
 and perform transposition with given matrix by saving the result into the Matrix named result*/
void Matrix_Transpose(const Matrix& matrix, Matrix&result){

    //  Matrix_Allocate function allocates memory for result.
    Matrix_Allocate(result, matrix.columnSize, matrix.rowSize);

    //
    for(int i=0; i<matrix.rowSize; i++){
        for(int j=0; j<matrix.columnSize; j++)
            result.data[j][i] = matrix.data[i][j];
    }//end FOR
}//end Matrix_Transpoze ()

//  Matrix_Row_Module takes two references(one is const) that their types are struct called Matrix.
/***  This function will call Matrix_Allocate function to create result array for matrix.rowSize x 1 sizes
 and perform row module with given matrix by saving the result into the Matrix named result*/
void Matrix_Row_Module(const Matrix& matrix, Matrix&result){

    //  Matrix_Allocate function allocates memory for result (matrix.rowSize x 1)
    Matrix_Allocate(result, matrix.rowSize, 1);

    for(int row=0, total=0; row<matrix.rowSize; row++){
        for(int col=0; col<matrix.columnSize; col++){
            total += pow(matrix.data[row][col],2);
        }//  Each element at the row is squared and totalled.

        result.data[row][0] = sqrt(total);
        //The square root of the total is assigned to result.data[row][0]

    }//end FOR

}//end Matrix_Row_Module ()

//  Matrix_Column_Module takes two references(one is const) that their types are struct called Matrix.
/***  This function will call Matrix_Allocate function to create result array for 1 x matrix.columnSize sizes
 and perform column module with given matrix by saving the result into the Matrix named result*/
void Matrix_Column_Module(const Matrix& matrix, Matrix&result){

    //  Matrix_Allocate function allocates memory for result (1 x matrix.columnSize)
    Matrix_Allocate(result, 1, matrix.columnSize);

    for(int col=0, total=0; col<matrix.columnSize; col++){
        for(int row=0; row<matrix.rowSize; row++){
            total += pow(matrix.data[row][col],2);
        }//  Each element at the column is squared and totalled.

        result.data[0][col] = sqrt(total);
        //The square root of the total is assigned to result.data[0][col]

    }//end FOR

}//end Matrix_Column_Module ()
```

```
/****************************************
 * MatrixTestApp.cpp                    *
 ****************************************
 * IDE : Xcode                          *
```

```
 * Author : Şafak AKINCI                *
 * Experiment 2: Introduction to C++ - II*
 ****************************************/

//  Created a test file which includes the entry point of the program (main)
//  and its name is MatrixTestApp.cpp and copied the given test code to it.

#include "Matrix.h"                //Adding Matrix.h header file that include function prototypes.
#include <iostream>                //Adding iostream header to use standart input output functions.
#include <string>                  //Adding string header to find string's length via length () function.
#include <iomanip>                 //Adding iomanip header to use setw() function but didn't use.

using namespace std;               //To don't write for each code std::   (e.x. std::cout)

//  PrintFrameLine function prints +-----+ according to the length.
void PrintFrameLine(int length);
//  PrintMessageInFrame function prints the message to console.
void PrintMessageInFrame(const string& message);
//  GetRandomData will create two-dimensional array and fill it with random numbers.
float** GetRandomData(int row, int column);

void TEST_FILL_BY_VALUE();
void TEST_FILL_BY_DATA();
void TEST_ADDITION();
void TEST_SUBSTRUCTION();
void TEST_MULTIPLICATION_MATRIX();
void TEST_MULTIPLICATION_CONSTANT();
void TEST_DIVISION();
void TEST_QUIZ();

int main() {
    TEST_FILL_BY_VALUE();
    TEST_FILL_BY_DATA();
    TEST_ADDITION();
    TEST_SUBSTRUCTION();
    TEST_MULTIPLICATION_MATRIX();
    TEST_MULTIPLICATION_CONSTANT();
    TEST_DIVISION();
    TEST_QUIZ();

    return 0;
}//end main()

//  PrintFrameLine function takes one integer parameter called length(message.length) and prints +-----+
void PrintFrameLine (int length){

    cout << "+";
    length -= 2;

    for (int i = 0; i < length; i++)
    {
        cout << "-";
    }

    cout << "+" << endl;
}//end PrintFrameLine ()

//  PrintMessageInFrame function takes one const string reference called message and prints it to console.
void PrintMessageInFrame (const string& message ){

    //  Added (unsigned int) in front of the message.length() function to get integer (lost precision).
    PrintFrameLine( (unsigned int)message.length() + 4 );
    cout << "| " << message << " |" << endl;
```

```
    PrintFrameLine( message.length() + 4 );
    //If we don't add (unsigned int) in front of the message.length() function Xcode warns the developer with
    //  " Implicit conversion loses integer precision: 'unsigned long' to 'int' ".

}//end PrintMessageInFrame ()

//  GetRandomData function takes two integer parameters and will create two-dimensional array and return it.
float** GetRandomData(int row, int column){

    //  row times elements that all of them are float(float*) are created in HEAP MEMORY,
    //  and assigned to matrixData.
    float** matrixData = new float*[row];

//To get two-dimensional array, each ROW of the array must have columnSize times elements(float).
    for (int i = 0; i < row; i++){
        matrixData[i] = new float[column];
    }

    //  Random numbers are assigned to matrixData.
    for (int i = 0; i < row; i++){
        for(int j = 0; j < column; j++){
            matrixData[i][j] = -10 + rand() % (22);
        }//end for
    }//end FOR


    //matrixData is returned to where it is called.
    return matrixData;
}//end GetRandomData ()

void TEST_FILL_BY_VALUE()
{
//  Print "FILL BY VALUE TEST" to console.
    PrintMessageInFrame("FILL BY VALUE TEST");

//  m1 is created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  All of the m1's elements will assign to 1.34 .
    Matrix_FillByValue(m1, 1.34);

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 4, 3);

//  All of the m1's elements will assign to -2.65 .
    Matrix_FillByValue(m1, -2.65);

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
```

```
}//end TEST_FILL_BY_VALUE ()

void TEST_FILL_BY_DATA(){

//  Print "FILL BY DATA SET" to console.
    PrintMessageInFrame("FILL BY DATA TEST");

//  m1 is created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 4, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(4, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);

}//end TEST_FILL_BY_DATA ()

void TEST_ADDITION(){

//  Print "ADDITION TEST" to console.
    PrintMessageInFrame("ADDITION TEST");

//  m1, m2 and m3 are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, m2, m3;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));
    cout << "First Matrix:" << endl;

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Allocate function will allocate memory for m2.
    Matrix_Allocate(m2, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m2, GetRandomData(2, 3));

    cout << "Second Matrix:" << endl;

//  All of the m2's elements will print to console.
```

```
    Matrix_Display(m2);

//  Matrix_Addition function will add m1 to m2 and save the result to m3.
    Matrix_Addition(m1, m2, m3);
    cout << "Result Matrix:" << endl;

//  All of the m3's elements will print to console.
    Matrix_Display(m3);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(m2);
    Matrix_Free(m3);

}//end TEST_ADDITION

void TEST_SUBSTRUCTION(){

//  Print "SUBSTRUCTION TEST" to console.
    PrintMessageInFrame("SUBSTRUCTION TEST");

//  m1, m2 and m3 are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, m2, m3;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));
    cout << "First Matrix:" << endl;

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Allocate function will allocate memory for m2.
    Matrix_Allocate(m2, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m2, GetRandomData(2, 3));

    cout << "Second Matrix:" << endl;

//  All of the m2's elements will print to console.
    Matrix_Display(m2);

//  Matrix_Substruction function will subtract m2 from m1 and save the result to m3.
    Matrix_Substruction(m1, m2, m3);

    cout << "Result Matrix:" << endl;

//  All of the m3's elements will print to console.
    Matrix_Display(m3);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(m2);
    Matrix_Free(m3);
}//end TEST_SUBSTRUCTION

void TEST_MULTIPLICATION_MATRIX (){

//  Print "MATRIX MULTIPLICATION TEST" to console.
    PrintMessageInFrame("MATRIX MULTIPLICATION TEST");
```

```
//  m1, m2 and m3 are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, m2, m3;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

    cout << "First Matrix:" << endl;

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Allocate function will allocate memory for m2.
    Matrix_Allocate(m2, 3, 2);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m2, GetRandomData(3, 2));

    cout << "Second Matrix:" << endl;

//  All of the m2's elements will print to console.
    Matrix_Display(m2);

//  Matrix_Multiplication function will multiply m1 with m2 and save the result to m3.
    Matrix_Multiplication(m1, m2, m3);

    cout << "Result Matrix:" << endl;

//  All of the m3's elements will print to console.
    Matrix_Display(m3);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(m2);
    Matrix_Free(m3);
}//end TEST_MULTIPLICATION_MATRIX

void TEST_MULTIPLICATION_CONSTANT(){

//  Print "SCALAR MULTIPLICATION TEST" to console.
    PrintMessageInFrame("SCALAR MULTIPLICATION TEST");

//  m1 and m2 are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, m2;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Multiplication function will multiply m1 with scalar and save the result to m2.
    float scalar = 3;
    Matrix_Multiplication(m1, scalar, m2);

    cout << "Result Matrix:" << endl;
```

```
//  All of the m2's elements will print to console.
    Matrix_Display(m2);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(m2);
}//end TEST_MULTIPLICATION_CONSTANT

void TEST_DIVISION (){

//  Print "SCALAR DIVISION TEST" to console.
    PrintMessageInFrame("SCALAR DIVISION TEST");

//  m1 and m2 are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, m2;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Division function will divide m1 to scalar and save the result to m2.
    float scalar = 3;
    Matrix_Division(m1, scalar, m2);

    cout << "Result Matrix:" << endl;

//  All of the m2's elements will print to console.
    Matrix_Display(m2);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(m2);
}//end TEST_DIVISION

void TEST_QUIZ(){

//  Print "TRANSPOSE TEST" to console.
    PrintMessageInFrame("TRANSPOSE TEST");

//  m1 and result are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, result;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Transpose function transpozes the given matrix and saves it to result.
    Matrix_Transpose(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);

//  Print "ROW MODULE TEST" to console.
```

```
    PrintMessageInFrame("ROW MODULE TEST");

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Row_Module function calculate the row module and saves it to result.
    Matrix_Row_Module(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);

//  Print "COLUMN MODULE TEST" to console.
    PrintMessageInFrame("COLUMN MODULE TEST");

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Column_Module function calculate the column module and saves it to result.
    Matrix_Column_Module(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(result);
}
```

Quiz

```
//  Transposing the given matrix and saving it to result matrix.
void Matrix_Transpose(const Matrix& matrix, Matrix&result);

//  Calculating the row module of the given matrix and saving it to result matrix.
void Matrix_Row_Module(const Matrix& matrix, Matrix&result);

//  Calculating the column module of the given matrix and saving it to result matrix.
void Matrix_Column_Module(const Matrix& matrix, Matrix&result);



void TEST_QUIZ(){

//  Print "TRANSPOSE TEST" to console.
    PrintMessageInFrame("TRANSPOSE TEST");

//  m1 and result are created the type of the struct Matrix which is defined in Matrix.h.
    Matrix m1, result;

//  Matrix_Allocate function will allocate memory for m1.
    Matrix_Allocate(m1, 2, 3);

//  Matrix_FillByData function will fill the given matrix with random numbers.
    Matrix_FillByData(m1, GetRandomData(2, 3));

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Transpose function transpozes the given matrix and saves it to result.
    Matrix_Transpose(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);
```

```
//  Print "ROW MODULE TEST" to console.
    PrintMessageInFrame("ROW MODULE TEST");

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Row_Module function calculate the row module and saves it to result.
    Matrix_Row_Module(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);

//  Print "COLUMN MODULE TEST" to console.
    PrintMessageInFrame("COLUMN MODULE TEST");

//  All of the m1's elements will print to console.
    Matrix_Display(m1);

//  Matrix_Column_Module function calculate the column module and saves it to result.
    Matrix_Column_Module(m1, result);

//  All of the m1's elements will print to console.
    Matrix_Display(result);

//  Matrix_Free function will delete the elements of the given parameter.
    Matrix_Free(m1);
    Matrix_Free(result);
}
```

Conclusion

In this lab I've learnt those things:

*How to create header file and why I should it.

***header** file is used for declaring function prototypes.

***const** prefix prevents changing given parameter in function.

***reference** is used to achieve given parameter under the name of the function's parameter, but there won't be another variable for function, just will use the given parameter.