Şafak AKINCI **08.11.2016**
152120141003

Experiment 3
Class vs. Struct

Objectives

To convert a struct based library into a class based library in C++.

Prelab Activities
Lab Exercise 1 – Matrix Library Creation (Class based)

```
/****************************************
 * Matrix.h                             *
 ****************************************
 * IDE : Xcode                          *
 * Author : Şafak AKINCI                *
 * Experiment 3: Class vs Struct        *
 *****************************************/

/*
 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.
 */
#pragma once


class Matrix
{

//Public members are accessible from anywhere where the object is visible.
public:
    //Those are member functions. They can call for each member that is created from Matrix class.

    //Allocates memory for required sizes to object.
    void Allocate(int rSize, int cSize);

    //The memory which was allocated for object will be free.
    void Free();

    //Gets row size of the object for another objects since rowSize is private.
    //The returned value the type of integer will be constant so that it doesn't change.
```

```
int getRowSize() const;

//Assigns given parameter to object's rowSize.
void setRowSize(int rSize);

//Gets column size of the object for another objects since columnSize is private.
//The returned value the type of integer will be constant so that it doesn't change.
int getColumnSize() const;

//Assigns given parameter to object's columnSize.
void setColumnSize(int cSize);

//Gets data of the object for another objects since data is private.
//The returned value the type of float to float pointer will be constant so that it doesn't change.
float** getData() const;

//Assigns given parameter(two-dim array) to object.
void setData(float** d);

//Fills the object with given value.
void FillByValue(float value);

//Assigns the given data to object's data.
void FillByData(float** d);

//Display the object's items.
void Display();

//Adds given object that is created from Marix class to object.
Matrix Addition(const Matrix& matrix_right);

//Subtracts given object that is created from Marix class from object.
Matrix Substruction(const Matrix& matrix_right);

//Multiplies given object that is created from Marix class with object.
Matrix Multiplication(const Matrix& matrix_right);

//Multiplies given scalarValue with object.
Matrix Multiplication(float scalarValue);

//Divides object to scalarValue.
Matrix Division(float scalarValue);

//Transposes the object.
Matrix Transpose();

//Calculates the row module of the object.
Matrix Row_Module();
```

```
    //Calculates the column module of the object.
    Matrix Column_Module();

//Private members of a class are accessible only from within other members of the same class.
private:
    int rowSize;
    int columnSize;
    float** data;
};
```

```
/*****************************************
 * Matrix.cpp                            *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 3: Class vs Struct         *
 *****************************************/

#include "Matrix.h"                         //Adding Matrix.h header file that include function prototypes.
#include <math.h>                           //To use pow() and sqrt() functions.
#include <iostream>                         //To use standart input output functions.
#include <iomanip>                          //To use setw() function but didn't use.
using namespace std;                        //To don't write for each code std::   (e.x. std::cout)

//  Allocates two-dimensional dynamic array into the data member of the Matrix
//  and updates rowSize and columnSize variables.
void Matrix::Allocate(int rSize, int cSize)
{
    //setRowSize(rSize);
    rowSize = rSize;

    //setColumnSize(cSize);
    columnSize = cSize;


    //  data is a pointer to pointer(float**) which is created in Matrix class will point another pointer.

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

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

}//end Matrix::Allocate ()
```

```cpp
//  Releases the allocated memory for the given Matrix and assigns rowSize and columnSize to -1 and data to nullptr.
void Matrix::Free()
{

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

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

    //setRowSize(-1);
    rowSize = -1;

    //setColumnSize(-1);
    columnSize = -1;

    data = nullptr;
}//end Matrix::Free ()

//Gets row size of the object for another objects since rowSize is private.
//The returned value the type of integer will be constant so that it doesn't change.
int Matrix::getRowSize() const
{
    int rSize;

    rSize = rowSize;

    return rSize;
}//end Matrix::getRowSize ()

//Assigns given parameter to object's rowSize.
void Matrix::setRowSize(int rSize)
{
    rowSize = rSize;
}//end Matrix::setRowSize ()

//Gets column size of the object for another objects since columnSize is private.
//The returned value the type of integer will be constant so that it doesn't change.
int Matrix::getColumnSize() const
{
    int cSize;

    cSize = columnSize;

    return columnSize;
}//end Matrix::getColumnSize()
```

```
//Assigns given parameter to object's columnSize.
void Matrix::setColumnSize(int cSize)
{
    columnSize = cSize;
}//end Matrix::setColumnSize ()

//Gets data of the object for another objects since data is private.
//The returned value the type of float to float pointer will be constant so that it doesn't change.
float** Matrix::getData() const
{
    return data;
}//end Matrix::getData ()

//Assigns given parameter(two-dim array) to object's data.
void Matrix::setData(float** d)
{
    data = d;
}//end Matrix::setData ()

//Fills the data member of the Matrix by the given value.
void Matrix::FillByValue(float value)
{

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

}//end Matrix::FillByValue ()

//Fills the data member of the Matrix by the corresponding elements of the given two-dimensional array.
void Matrix::FillByData(float** d)
{

    //data = d;

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

}//Matrix::FillByData ()

//Displays the Matrix.
```

```cpp
void Matrix::Display()
{

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

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

}//end Matrix::Display ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs matrix addition.
Matrix Matrix::Addition(const Matrix& matrix_right)
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, columnSize);

    //Two matrix is added and the result is assigned to result object.
    for(int i=0; i<rowSize; i++){
        for(int j=0; j<columnSize; j++){
            result.data[i][j] = data[i][j] + matrix_right.data[i][j];
        }//end for
    }//end FOR


    return result;

}//end Matrix::Addition ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs matrix subtraction.
Matrix Matrix::Substruction(const Matrix& matrix_right)
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, columnSize);

    //Given object's data is subtracted from object's data and result is assigned result's data.
    for(int i=0; i<rowSize; i++){
        for(int j=0; j<columnSize; j++){
            result.data[i][j] = data[i][j] - matrix_right.data[i][j];
        }//end for
    }//end FOR

    return result;

}//end Matrix::Substruction ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs matrix multiplication.
Matrix Matrix::Multiplication(const Matrix& matrix_right)
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, matrix_right.columnSize);

    //Multiplies given object with object and result is assigned the result matrix.
    for(int i=0;i<rowSize;i++){
        for(int j=0;j<columnSize;j++){

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

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

        }//end FOR
    }//END FOR

    return result;

}//end Matrix::Multiplication ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs multiplication with
scalarValue.
Matrix Matrix::Multiplication(float scalarValue)
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, columnSize);

    //Multiplies given scalarValue with object and result is assigned the result matrix.
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = data[i][j] * scalarValue;
        }//end for
    }//end FOR
    return result;
}//end Matrix::Multiplication ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs division with 
scalarValue.
Matrix Matrix::Division(float scalarValue)
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, columnSize);

    //Divides object to given scalarValue and result is assigned the result matrix.
    for(int i=0; i<result.rowSize; i++){
        for(int j=0; j<result.columnSize; j++){
            result.data[i][j] = data[i][j] / scalarValue;
        }//end for
    }//end FOR

    return result;
}//end Matrix::Division ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs transpose operation.
Matrix Matrix::Transpose()
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(columnSize, rowSize);

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

    return result;
}//end Matrix::Transpose ()

//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs matrix row module.
Matrix Matrix::Row_Module()
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(rowSize, 1);
    for(int row=0, total=0; row<rowSize; row++){
        for(int col=0; col<columnSize; col++){
            total += pow(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

    return result;

}//end Matrix::Row_Module ()


//Creates a new matrix and calls Allocate function for the result Matrix for required sizes and performs matrix column module.
Matrix Matrix::Column_Module()
{
    //result object is created by Matrix class.
    Matrix result;

    //Allocates memory for required sizes to it.
    result.Allocate(columnSize, 1);

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

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

    }//end FOR

    return result;

}//end Matrix::Column_Module ()
```

```
/*****************************************
 * MatrixTestApp.cpp                     *
 *****************************************
 * IDE : Xcode                           *
 * Author : Şafak AKINCI                 *
 * Experiment 3: Class vs Struct         *
 *****************************************/

#include "Matrix.h"                        //Adding Matrix.h header file that include function prototypes.
#include <iostream>                        //To use standart input output functions.
#include <string>                          //To create string variable.
#include <iomanip>                         //To use setw() function but didn't use.
using namespace std;                       //To don't write for each code std::   (e.x. std::cout)

//  Prints +----+ according to the length.
void PrintFrameLine(int length);

//  Prints the message to console.
void PrintMessageInFrame(const string& message);

//  Creates two-dimensional array and fills 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_TRANSPOSE();
void TEST_ROW_MODULE();
void TEST_COLUMN_MODULE();

int main() {
    TEST_FILL_BY_VALUE();
    TEST_FILL_BY_DATA();
    TEST_ADDITION();
    TEST_SUBSTRUCTION();
    TEST_MULTIPLICATION_MATRIX();
    TEST_MULTIPLICATION_CONSTANT();
    TEST_DIVISION();
    TEST_TRANSPOSE();
    TEST_ROW_MODULE();
    TEST_COLUMN_MODULE();

    return 0;
}//end main ()

// Takes one integer parameter called length(message.length) and prints +----+ according to it.
void PrintFrameLine(int length){
    cout << "+";
    length -= 2;
    for (int i = 0; i < length; i++)
    {
```

```cpp
            cout << "-";
        }

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

    // 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)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 ()

    // Takes two integer parameters and creates two-dimensional array and returns 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 object is created from Matrix class.     (It has members and member functions.)
        Matrix m1;

        //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
        m1.Allocate(2, 3);

        //  FillByValue function which is the member function of m1 is called to fill the matrix with given value.
        m1.FillByValue(1.34);

        //  Display function which is the member function of m1 is called to display its items.
        m1.Display();

        //  Free function which is the member function of m1 is called to free the memory that was allocated for m1.
        m1.Free();

        //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
        m1.Allocate(4, 3);

        //  FillByValue function which is the member function of m1 is called to fill the matrix with given value.
        m1.FillByValue(-2.65);

        //  Display function which is the member function of m1 is called to display its items.
        m1.Display();

        //  Free function which is the member function of m1 is called to free the memory that was allocated for m1.
        m1.Free();

    }//end TEST_FILL_BY_VALUE ()

    void TEST_FILL_BY_DATA()
    {
        //  Print "FILL BY DATA TEST" to console.
        PrintMessageInFrame("FILL BY DATA TEST");

        //  m1 object is created from Matrix class.     (It has members and member functions.)
        Matrix m1;

        //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
        m1.Allocate(2, 3);

        //  FillByData function which is the member function of m1 is called to fill the matrix with random data
        //  that is generated GetRandomData function.
        m1.FillByData(GetRandomData(2, 3));

        //  Display function which is the member function of m1 is called to display its items.
        m1.Display();

        //  Free function which is the member function of m1 is called to free the memory that was allocated for m1.
        m1.Free();
```

```cpp
    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(4, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(4, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  Free function which is the member function of m1 is called to free the memory that was allocated for m1.
    m1.Free();

}//end TEST_FILL_BY_DATA ()

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

    //  Objects are created from matrix class.
    Matrix m1, m2, m3;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

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

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  Allocate function which is the member function of m2 is called to allocate memory for required sizes.
    m2.Allocate(2, 3);

    //  FillByData function which is the member function of m2 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m2.FillByData(GetRandomData(2, 3));

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  m1 and m2 is added and assigned to m3.
    m3 = m1.Addition(m2);
    cout << "Result Matrix:" << endl;

    //  Display function which is the member function of m3 is called to display its items.
    m3.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();
    m3.Free();
}//end TEST_ADDITION ()

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

    //  Objects are created from matrix class.
    Matrix m1, m2, m3;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

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

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  Allocate function which is the member function of m2 is called to allocate memory for required sizes.
    m2.Allocate(2, 3);

    //  FillByData function which is the member function of m2 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m2.FillByData(GetRandomData(2, 3));

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  m2 is subtracted from m1 and the result is assigned to m3.
    m3 = m1.Substruction(m2);

    cout << "Result Matrix:" << endl;
    //  Display function which is the member function of m3 is called to display its items.
    m3.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();
    m3.Free();

}//end TEST_SUBSTRUCTION ()

void TEST_MULTIPLICATION_MATRIX()
{
    //  Print "MATRIX MULTIPLICATION TEST" to console.
    PrintMessageInFrame("MATRIX MULTIPLICATION TEST");

    //  Objects are created from matrix class.
    Matrix m1, m2, m3;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

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

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  Allocate function which is the member function of m2 is called to allocate memory for required sizes.
    m2.Allocate(3, 2);


    //  FillByData function which is the member function of m2 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m2.FillByData(GetRandomData(3, 2));

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  m1 and m2 is multiplied and the result is assigned to m3.
    m3 = m1.Multiplication(m2);

    cout << "Result Matrix:" << endl;
    //  Display function which is the member function of m3 is called to display its items.
    m3.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();
    m3.Free();

}//end TEST_MULTIPLICATION_MATRIX ()

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

    //  Objects are created from matrix class.
    Matrix m1, m2;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    float scalar = 3;
    //  m1 is divided to given scalar value and the result is assigned to m2.
    m2 = m1.Multiplication(scalar);

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();

}//end TEST_MULTIPLICATION_CONSTANT ()

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

    //  Objects are created from matrix class.
    Matrix m1, m2;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    float scalar = 3;
    //  m1 is divided to given scalar value and the result is assigned to m2.
    m2 = m1.Division(scalar);

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();

}//end TEST_DIVISION ()

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

    //  Objects are created from matrix class.
    Matrix m1, m2;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();
    //  m1's transpose is assigned to m2.
    m2 = m1.Transpose();

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();

}//end TEST_TRANSPOSE ()

void TEST_ROW_MODULE()
{
    //  Print "ROW MODULE TEST" to console.
    PrintMessageInFrame("ROW MODULE TEST");

    //  Objects are created from matrix class.
    Matrix m1, m2;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData(GetRandomData(2, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  m1's row module is calculated and the result is assigned to m2.
    m2 = m1.Row_Module();

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();

}//end TEST_ROW_MODULE ()
```

```cpp
void TEST_COLUMN_MODULE()
{
    //  Print "COLUMN MODULE TEST" to console.
    PrintMessageInFrame("COLUMN MODULE TEST");

    //  Objects are created from matrix class.
    Matrix m1, m2;

    //  Allocate function which is the member function of m1 is called to allocate memory for required sizes.
    m1.Allocate(2, 3);

    //  FillByData function which is the member function of m1 is called to fill the matrix with random data
    //  that is generated GetRandomData function.
    m1.FillByData( GetRandomData(2, 3));

    //  Display function which is the member function of m1 is called to display its items.
    m1.Display();

    //  m1's column module is calculated and the result is assigned to m2.
    m2 = m1.Column_Module();

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

    //  Display function which is the member function of m2 is called to display its items.
    m2.Display();

    //  Free function which is the member function of each object is called to free the memory that was allocated for each of
them.
    m1.Free();
    m2.Free();

}//end TEST_COLUMN_MODULE ()
```

Conclusion

*   Classes can contain data members, but they can also contain functions as members.
*   Accessing those members can be limited or not via access specifiers called private and public.
*   A private members of an object can be accessed by using object's member function like getData.
*   Function’s returned value can be set as constant to don’t change its data.