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)

The problem is divided into five parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template
5. Problem-Solving Tips

The program template represents a complete working C++ program. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, implement the C++ code required. Compile and execute the program. Compare your output with the sample output provided.

Lab Objectives

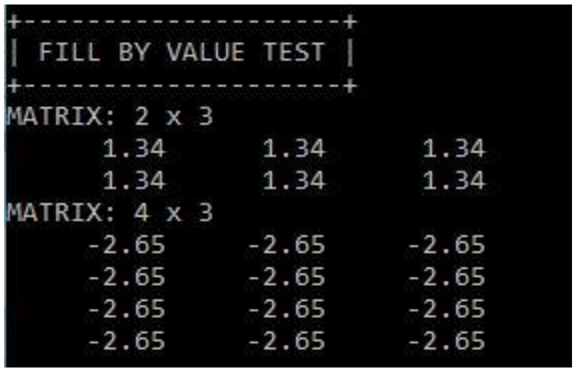
In this lab, you will practice:

- Using cout to output text and variables.
- Using cin to input data from the user.
- Using the arithmetic operators to perform calculations.
- Using two dimensional arrays to hold the data
- Using loops to calculate some operations
- Using C++ references to reference a variable
- Using classes
- Using I/O manipulation to visualize output in a proper way

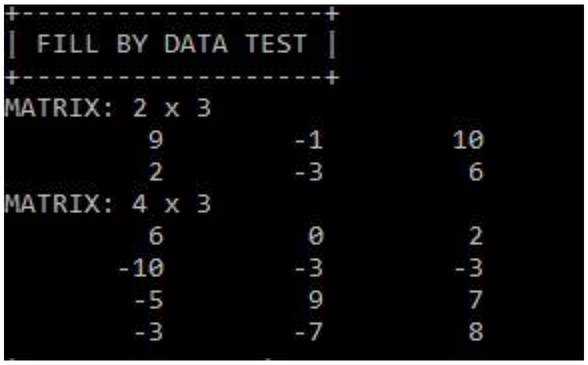
Description of the Problem

Write a C++ Matrix library (class based) and a test program that performs the basic matrix operations by using the given class type (Matrix). Create a header file which is named by Matrix.h and copy the given function prototypes. Then create another file which is name by Matrix.cpp and include the Matrix.h in this file and implement the functions located in the file Matrix.h. Create a test file which includes the entry point of your program (main) and give a name to that file (MatrixTestApp.cpp) and copy the given test code into that file. Compare the results obtained and the given sample code. (Note: You are not allowed to change the given function prototypes or add a new function, and also given test code. Just implement required functions).

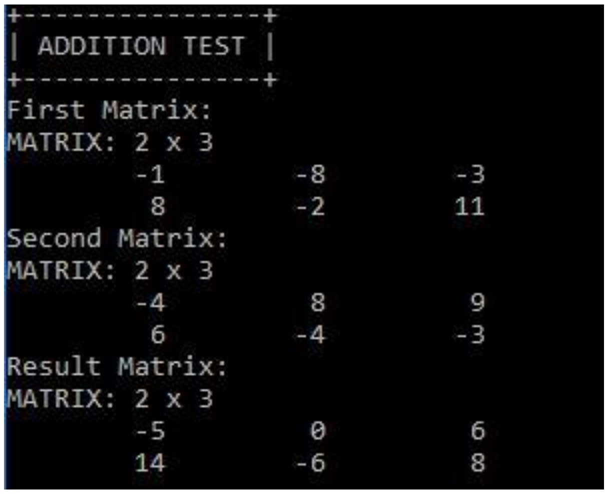
Sample Code



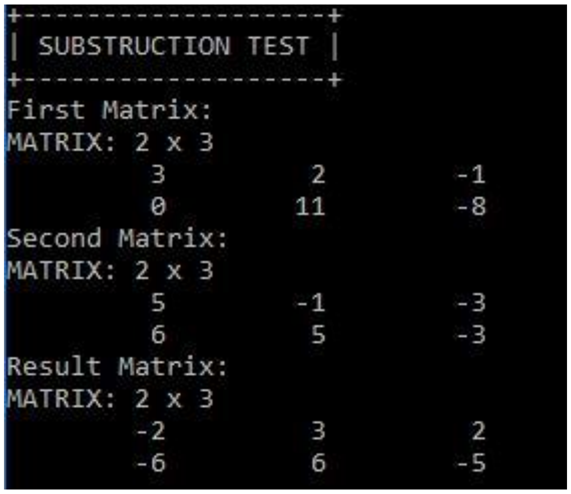
```
+--------------------+
| FILL BY VALUE TEST |
+--------------------+
MATRIX: 2 x 3
      1.34      1.34      1.34
      1.34      1.34      1.34
MATRIX: 4 x 3
     -2.65     -2.65     -2.65
     -2.65     -2.65     -2.65
     -2.65     -2.65     -2.65
     -2.65     -2.65     -2.65
```



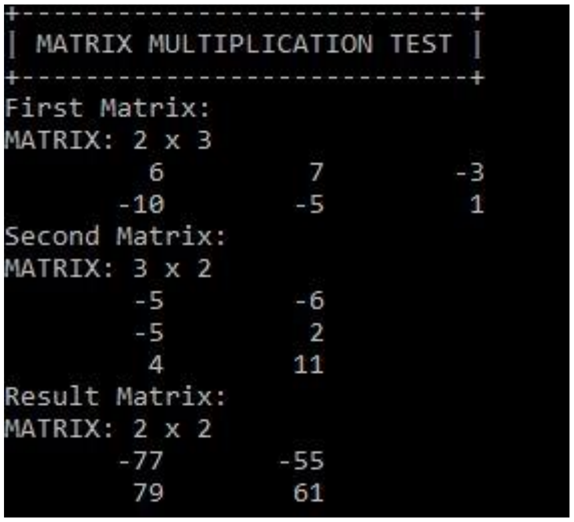
```
+-------------------+
| FILL BY DATA TEST |
+-------------------+
MATRIX: 2 x 3
         9        -1        10
         2        -3         6
MATRIX: 4 x 3
         6         0         2
       -10        -3        -3
        -5         9         7
        -3        -7         8
```



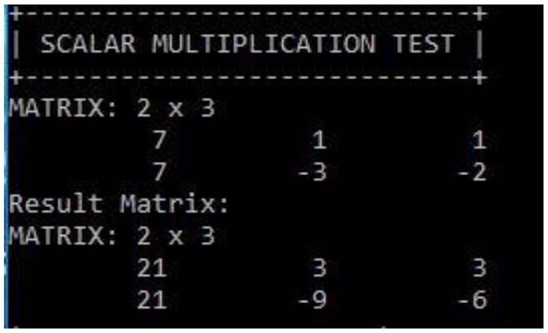
```
+---------------+
| ADDITION TEST |
+---------------+
First Matrix:
MATRIX: 2 x 3
        -1        -8        -3
         8        -2        11
Second Matrix:
MATRIX: 2 x 3
        -4         8         9
         6        -4        -3
Result Matrix:
MATRIX: 2 x 3
        -5         0         6
        14        -6         8
```



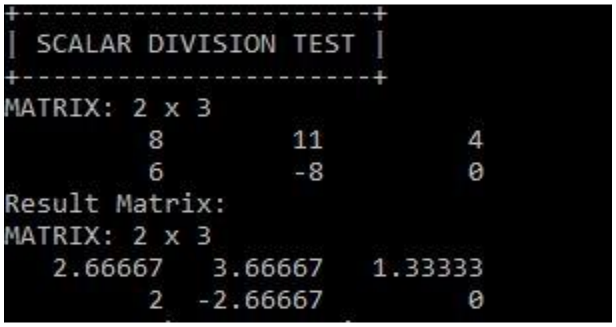
```
+-------------------+
| SUBSTRUCTION TEST |
+-------------------+
First Matrix:
MATRIX: 2 x 3
         3         2        -1
         0        11        -8
Second Matrix:
MATRIX: 2 x 3
         5        -1        -3
         6         5        -3
Result Matrix:
MATRIX: 2 x 3
        -2         3         2
        -6         6        -5
```



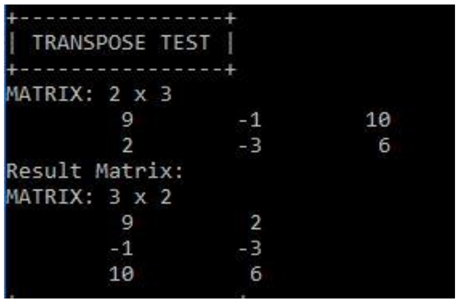
```
+----------------------------+
| MATRIX MULTIPLICATION TEST |
+----------------------------+
First Matrix:
MATRIX: 2 x 3
         6         7        -3
       -10        -5         1
Second Matrix:
MATRIX: 3 x 2
        -5        -6
        -5         2
         4        11
Result Matrix:
MATRIX: 2 x 2
       -77       -55
        79        61
```



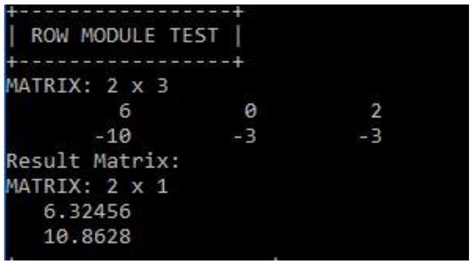
```
+----------------------------+
| SCALAR MULTIPLICATION TEST |
+----------------------------+
MATRIX: 2 x 3
         7         1         1
         7        -3        -2
Result Matrix:
MATRIX: 2 x 3
        21         3         3
        21        -9        -6
```



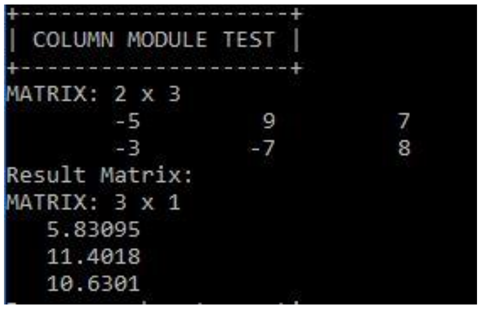
```
+----------------------+
| SCALAR DIVISION TEST |
+----------------------+
MATRIX: 2 x 3
         8        11         4
         6        -8         0
Result Matrix:
MATRIX: 2 x 3
   2.66667   3.66667   1.33333
         2  -2.66667         0
```



```
+---------------+
| TRANSPOSE TEST |
+---------------+
MATRIX: 2 x 3
         9          -1          10
         2          -3           6
Result Matrix:
MATRIX: 3 x 2
         9           2
        -1          -3
        10           6
```



```
+-----------------+
| ROW MODULE TEST |
+-----------------+
MATRIX: 2 x 3
         6           0           2
       -10          -3          -3
Result Matrix:
MATRIX: 2 x 1
   6.32456
   10.8628
```



```
+--------------------+
| COLUMN MODULE TEST |
+--------------------+
MATRIX: 2 x 3
        -5           9           7
        -3          -7           8
Result Matrix:
MATRIX: 3 x 1
   5.83095
   11.4018
   10.6301
```

Code Template

```
/*****************************************
* Matrix.h                               *
*****************************************
* IDE : Visual Studio 2015               *
* Author : Cihan UYANIK                  *
* Experiment 3: Class vs Struct          *
*****************************************/
#pragma once
class Matrix
{
public:
	void Allocate(int rSize, int cSize);
	void Free();

	int getRowSize() const;
	void setRowSize(int rSize);
	int getColumnSize() const;
	void setColumnSize(int cSize);
	float** getData() const;
	void setData(float** d);

	void FillByValue(float value);
	void FillByData(float** d);
	void Display();

	Matrix Addition(const Matrix& matrix_right);
	Matrix Substruction(const Matrix& matrix_right);
	Matrix Multiplication(const Matrix& matrix_right);
	Matrix Multiplication(float scalarValue);
	Matrix Division(float scalarValue);
	Matrix Transpose();
	Matrix Row_Module();
	Matrix Column_Module();

private:
	int rowSize;
	int columnSize;
	float** data;
};

/*****************************************
* Matrix.cpp                            *
*****************************************
* IDE : Visual Studio 2015              *
* Author : Cihan UYANIK                 *
* Experiment 3: Class vs Struct         *
*****************************************/
#include "Matrix.h"
#include <iostream>
#include <iomanip>
using namespace std;

void Matrix::Allocate(int rSize, int cSize)
{
// LOOK AT Problem Solving Tips Element (1)
}

void Matrix::Free()
{
// LOOK AT Problem Solving Tips Element (2)
}

int Matrix::getRowSize() const
{
}

void Matrix::setRowSize(int rSize)
{
}

int Matrix::getColumnSize() const
{
}

void Matrix::setColumnSize(int cSize)
{
}

float** Matrix::getData() const
{
}
```

```
void Matrix::setData(float** d)
{
}

void Matrix::FillByValue(float value)
{
// LOOK AT Problem Solving Tips Element (3)
}

void Matrix::FillByData(float** d)
{
// LOOK AT Problem Solving Tips Element (4)
}

void Matrix::Display()
{
// LOOK AT Problem Solving Tips Element (5)
}

Matrix Matrix::Addition(const Matrix& matrix_right)
{
// LOOK AT Problem Solving Tips Element (6)
}

Matrix Matrix::Substruction(const Matrix& matrix_right)
{
// LOOK AT Problem Solving Tips Element (7)
}

Matrix Matrix::Multiplication(const Matrix& matrix_right)
{
// LOOK AT Problem Solving Tips Element (8)
}

Matrix Matrix::Multiplication(float scalarValue)
{
// LOOK AT Problem Solving Tips Element (9)
}

Matrix Matrix::Division(float scalarValue)
{
// LOOK AT Problem Solving Tips Element (10)
}

Matrix Matrix::Transpose()
{
// LOOK AT Problem Solving Tips Element (11)
}

Matrix Matrix::Row_Module()
{
// LOOK AT Problem Solving Tips Element (12)
}

Matrix Matrix::Column_Module()
{
// LOOK AT Problem Solving Tips Element (13)
```

```
}

/*****************************************
* MatrixTestApp.cpp                      *
*****************************************
* IDE : Visual Studio 2015               *
* Author : Cihan UYANIK                  *
* Experiment 3: Class vs Struct          *
*****************************************/
#include "Matrix.h"
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

void PrintFrameLine(int length);
void PrintMessageInFrame(const string& message);
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;
}

void PrintFrameLine(int length)
{
       cout << "+";
       length -= 2;
       for (int i = 0; i < length; i++)
       {
              cout << "-";
       }

       cout << "+" << endl;
}

void PrintMessageInFrame(const string& message)
{
	PrintFrameLine(message.length() + 4);

	cout << "| " << message << " |" << endl;

	PrintFrameLine(message.length() + 4);
}

float** GetRandomData(int row, int column)
{
	float** matrixData = new float*[row];
	for (int i = 0; i < row; i++)
	{
		matrixData[i] = new float[column];
	}

	for (int i = 0; i < row; i++)
	{
		for (int j = 0; j < column; j++)
		{
			matrixData[i][j] = -10 + rand() % (22);
		}
	}

	return matrixData;
}

void TEST_FILL_BY_VALUE()
{
	PrintMessageInFrame("FILL BY VALUE TEST");

	Matrix m1;
	m1.Allocate(2, 3);
	m1.FillByValue(1.34);
	m1.Display();
	m1.Free();

	m1.Allocate(4, 3);
	m1.FillByValue(-2.65);
	m1.Display();
	m1.Free();
}

void TEST_FILL_BY_DATA()
{
	PrintMessageInFrame("FILL BY DATA TEST");

	Matrix m1;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	m1.Display();
	m1.Free();

	m1.Allocate(4, 3);
	m1.FillByData(GetRandomData(4, 3));
	m1.Display();
	m1.Free();
}
```

```
void TEST_ADDITION()
{
	PrintMessageInFrame("ADDITION TEST");

	Matrix m1, m2, m3;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	m1.Display();

	m2.Allocate(2, 3);
	m2.FillByData(GetRandomData(2, 3));
	cout << "Second Matrix:" << endl;
	m2.Display();

	m3 = m1.Addition(m2);
	cout << "Result Matrix:" << endl;
	m3.Display();

	m1.Free();
	m2.Free();
	m3.Free();
}

void TEST_SUBSTRUCTION()
{
	PrintMessageInFrame("SUBSTRUCTION TEST");

	Matrix m1, m2, m3;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	m1.Display();

	m2.Allocate(2, 3);
	m2.FillByData(GetRandomData(2, 3));
	cout << "Second Matrix:" << endl;
	m2.Display();

	m3 = m1.Substruction(m2);
	cout << "Result Matrix:" << endl;
	m3.Display();

	m1.Free();
	m2.Free();
	m3.Free();
}

void TEST_MULTIPLICATION_MATRIX()
{
	PrintMessageInFrame("MATRIX MULTIPLICATION TEST");

	Matrix m1, m2, m3;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	m1.Display();

	m2.Allocate(3, 2);
	m2.FillByData(GetRandomData(3, 2));
	cout << "Second Matrix:" << endl;
	m2.Display();

	m3 = m1.Multiplication(m2);
	cout << "Result Matrix:" << endl;
	m3.Display();

	m1.Free();
	m2.Free();
	m3.Free();
}

void TEST_MULTIPLICATION_CONSTANT()
{
	PrintMessageInFrame("SCALAR MULTIPLICATION TEST");
	Matrix m1, m2;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	m1.Display();

	float scalar = 3;

	m2 = m1.Multiplication(scalar);
	cout << "Result Matrix:" << endl;
	m2.Display();

	m1.Free();
	m2.Free();
}

void TEST_DIVISION()
{
	PrintMessageInFrame("SCALAR DIVISION TEST");
	Matrix m1, m2;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	m1.Display();

	float scalar = 3;

	m2 = m1.Division(scalar);
	cout << "Result Matrix:" << endl;
	m2.Display();

	m1.Free();
	m2.Free();
}

void TEST_TRANSPOSE()
{
	PrintMessageInFrame("TRANSPOSE TEST");
	Matrix m1, m2;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	m1.Display();

	m2 = m1.Transpose();
	cout << "Result Matrix:" << endl;
	m2.Display();

	m1.Free();
	m2.Free();
}

void TEST_ROW_MODULE()
{
	PrintMessageInFrame("ROW MODULE TEST");
	Matrix m1, m2;

	m1.Allocate(2, 3);
	m1.FillByData(GetRandomData(2, 3));
	m1.Display();

	m2 = m1.Row_Module();
	cout << "Result Matrix:" << endl;
	m2.Display();

	m1.Free();
	m2.Free();
}

void TEST_COLUMN_MODULE()
{
	PrintMessageInFrame("COLUMN MODULE TEST");
	Matrix m1, m2;

	m1.Allocate(2, 3);
	m1.FillByData( GetRandomData(2, 3));
	m1.Display();

	m2 = m1.Column_Module();
	cout << "Result Matrix:" << endl;
	m2.Display();

	m1.Free();
	m2.Free();
}
```

Problem Solving Tips

1. Allocate two-dimensional dynamic array into the `data` member of the `Matrix` and update `rowSize` and `columnSize` variables.
2. Free the allocated memory for the given `Matrix` and assign `rowSize` and `columnSize` to -1 and `data` to `nullptr` or 0.
3. Fill the `data` member of the `Matrix` by the given value

4. Fill the `data` member of the `Matrix` by the corresponding elements of the given two-dimensional array
5. Display the `Matrix` according to the given sample output
6. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix addition.
7. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix substruction.
8. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix multiplication.
9. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs scalar matrix multiplication (Multiply each element of the data member by the given constant value).
10. Similar to Tip (8) but for division.
11. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix transpose operation.
12. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix row module.

    Assume we have matrix named A with following coefficients.

a_{11}	a_{12}	a_{13}	…	a_{1m}
a_{21}	a_{22}	a_{23}	…	a_{2m}
a_{31}	a_{32}	a_{33}	…	a_{3m}
…	…	…	…	…
a_{n1}	a_{n2}	a_{n3}	…	a_{nm}

    It is required to calculate the row module of the matrix. The required row module matrix is can be calculated as following.

$\sqrt{a_{11}^2 + a_{12}^2 + a_{13}^2 + \cdots + a_{1m}^2}$
$\sqrt{a_{21}^2 + a_{22}^2 + a_{23}^2 + \cdots + a_{2m}^2}$
$\sqrt{a_{31}^2 + a_{32}^2 + a_{33}^2 + \cdots + a_{3m}^2}$
…
$\sqrt{a_{n1}^2 + a_{n2}^2 + a_{n3}^2 + \cdots + a_{nm}^2}$

13. Create a new matrix and call `Allocate` function for the result `Matrix` and for required sizes and performs matrix column module.

    For the same matrix given in the tip 12.

It is required to calculate the row module of the matrix. The required row module matrix is can be calculated as following.

$\sqrt{a_{11}^2 + a_{21}^2 + a_{31}^2 + \cdots + a_{n1}^2}$
$\sqrt{a_{12}^2 + a_{22}^2 + a_{32}^2 + \cdots + a_{n2}^2}$
$\sqrt{a_{13}^2 + a_{23}^2 + a_{33}^2 + \cdots + a_{n3}^2}$
…
$\sqrt{a_{1m}^2 + a_{2m}^2 + a_{3m}^2 + \cdots + a_{nm}^2}$