Experiment 2

Introduction to C++ Programming – II

Objectives

To write a simple computer programs in C++

Prelab Activities

Lab Exercise 1 – Matrix Library Creation

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

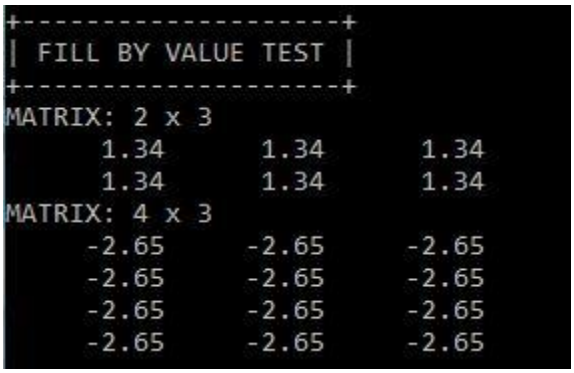
In this lab, you will practice:

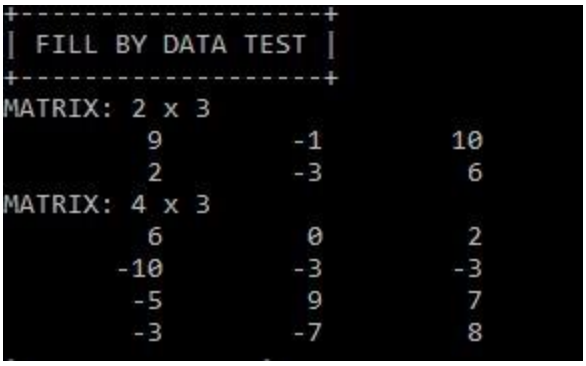
- Using cout to output text and variables.
- Using cin to input data from the user.
- Using the arithmetic operators to perform calculations.
- Using two dimensional arrays to hold the data
- Using loops to calculate some operations
- Using C++ references to reference a variable
- Using struct types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

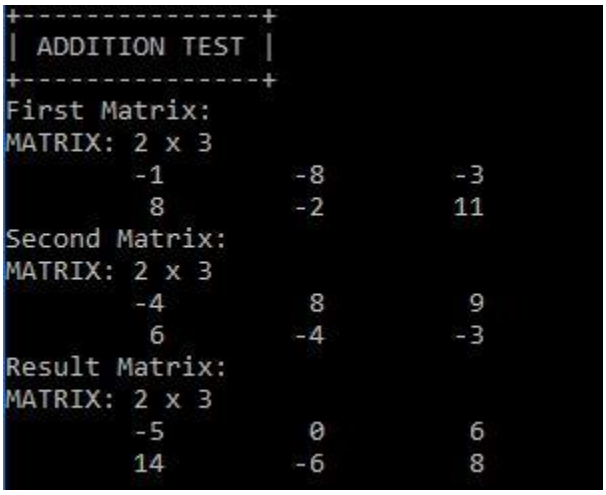
Description of the Problem

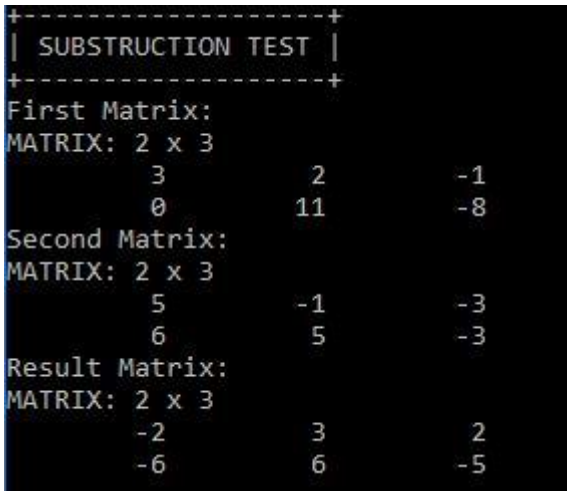
Write a C++ Matrix library and a test program that performs the basic matrix operations by using the given struct type (Matrix). Create a header file which is named by Matrix.h and copy the given function prototypes. Then create another file which is name by Matrix.cpp and include the Matrix.h in this 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).

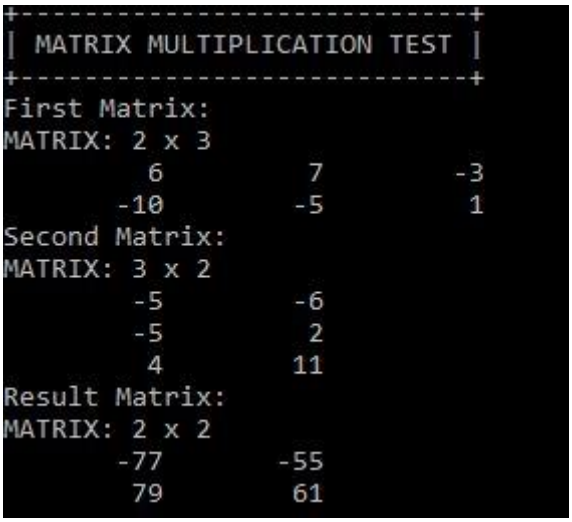
Sample Code

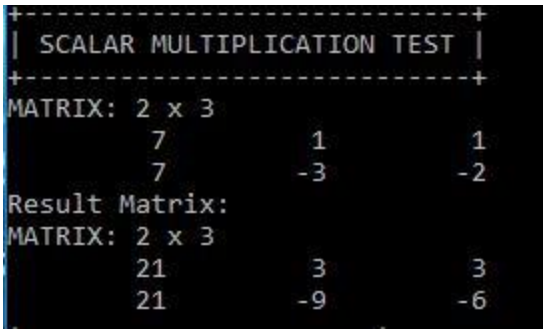

+--------------------+
| FILL BY VALUE TEST |
+--------------------+
MATRIX: 2 x 3
1.34 1.34 1.34
1.34 1.34 1.34
MATRIX: 4 x 3
-2.65 -2.65 -2.65
-2.65 -2.65 -2.65
-2.65 -2.65 -2.65
-2.65 -2.65 -2.65

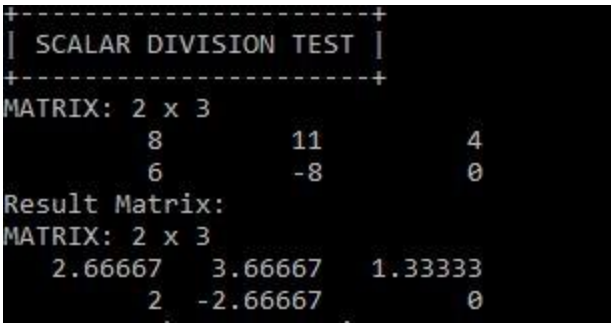

+-------------------+
| FILL BY DATA TEST |
+-------------------+
MATRIX: 2 x 3
9 -1 10
2 -3 6
MATRIX: 4 x 3
6 0 2
-10 -3 -3
-5 9 7
-3 -7 8


+---------------+
| 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

Code Template

```
/**************************************
* Matrix.h                            *
**************************************
* IDE : Visual Studio 2015            *
* Author : Cihan UYANIK               *
* Experiment 2: Introduction to C++ - II *
**************************************/
#pragma once

struct Matrix{
	int rowSize = -1;
	int columnSize = -1;
	float** data = 0;
};

void Matrix_Allocate(Matrix& matrix, int rowSize, int columnSize);
void Matrix_Free(Matrix& matrix);
void Matrix_FillByValue(Matrix& matrix, float value);
void Matrix_FillByData(Matrix& matrix, float** data);
void Matrix_Display(const Matrix& matrix);
void Matrix_Addition(const Matrix& matrix_left, const Matrix& matrix_right, Matrix&
result);
void Matrix_Substruction(const Matrix& matrix_left, const Matrix& matrix_right, Matrix&
result);
void Matrix_Multiplication(const Matrix& matrix_left, const Matrix& matrix_right, Matrix&
result);
void Matrix_Multiplication(const Matrix& matrix_left, float scalarValue, Matrix& result);
void Matrix_Division(const Matrix& matrix_left, float scalarValue, Matrix& result);



/**************************************
* Matrix.cpp                          *
**************************************
* IDE : Visual Studio 2015            *
* Author : Cihan UYANIK               *
* Experiment 2: Introduction to C++ - II *
**************************************/
#include  "Matrix.h"
#include  <iostream>
#include <iomanip>
using namespace std;


void Matrix_Allocate(Matrix& matrix, int rowSize, int columnSize)
{
// LOOK AT Problem Solving Tips Element (1)
}

void Matrix_Free(Matrix& matrix)
{
// LOOK AT Problem Solving Tips Element (2)
}

void Matrix_FillByValue(Matrix& matrix, float value)
{
```

```
// LOOK AT Problem Solving Tips Element (3)
}

void Matrix_FillByData(Matrix & matrix, float ** data)
{
// LOOK AT Problem Solving Tips Element (4)
}

void Matrix_Display(const Matrix& matrix)
{
// LOOK AT Problem Solving Tips Element (5)
}

void Matrix_Addition(const Matrix & matrix_left, const Matrix & matrix_right, Matrix &
result)
{
// LOOK AT Problem Solving Tips Element (6)
}

void Matrix_Substruction(const Matrix & matrix_left, const Matrix & matrix_right, Matrix &
result)
{
// LOOK AT Problem Solving Tips Element (7)
}

void Matrix_Multiplication(const Matrix & matrix_left, const Matrix & matrix_right, Matrix
& result)
{
// LOOK AT Problem Solving Tips Element (8)
}

void Matrix_Multiplication(const Matrix & matrix_left, float scalarValue, Matrix & result)
{
// LOOK AT Problem Solving Tips Element (9)
}

void Matrix_Division(const Matrix & matrix_left, float scalarValue, Matrix & result)
{
// LOOK AT Problem Solving Tips Element (10)
}


/**************************************
* MatrixTestApp.cpp                   *
**************************************
* IDE : Visual Studio 2015            *
* Author : Cihan UYANIK               *
* Experiment 2: Introduction to C++ - II *
**************************************/
#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();

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

	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByValue(m1, 1.34);
	Matrix_Display(m1);
	Matrix_Free(m1);


	Matrix_Allocate(m1, 4, 3);
	Matrix_FillByValue(m1, -2.65);
	Matrix_Display(m1);
	Matrix_Free(m1);
}

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

	Matrix m1;

	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	Matrix_Display(m1);
	Matrix_Free(m1);

	Matrix_Allocate(m1, 4, 3);
	Matrix_FillByData(m1, GetRandomData(4, 3));
	Matrix_Display(m1);
	Matrix_Free(m1);
}

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

	Matrix m1, m2, m3;

	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	Matrix_Display(m1);

	Matrix_Allocate(m2, 2, 3);
	Matrix_FillByData(m2, GetRandomData(2, 3));
	cout << "Second Matrix:" << endl;
	Matrix_Display(m2);

	Matrix_Addition(m1, m2, m3);
	cout << "Result Matrix:" << endl;
	Matrix_Display(m3);

	Matrix_Free(m1);
```

```
	Matrix_Free(m2);
	Matrix_Free(m3);
}

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

	Matrix m1, m2, m3;

	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	Matrix_Display(m1);

	Matrix_Allocate(m2, 2, 3);
	Matrix_FillByData(m2, GetRandomData(2, 3));
	cout << "Second Matrix:" << endl;
	Matrix_Display(m2);

	Matrix_Substruction(m1, m2, m3);
	cout << "Result Matrix:" << endl;
	Matrix_Display(m3);

	Matrix_Free(m1);
	Matrix_Free(m2);
	Matrix_Free(m3);
}

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

	Matrix m1, m2, m3;

	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	cout << "First Matrix:" << endl;
	Matrix_Display(m1);

	Matrix_Allocate(m2, 3, 2);
	Matrix_FillByData(m2, GetRandomData(3, 2));
	cout << "Second Matrix:" << endl;
	Matrix_Display(m2);

	Matrix_Multiplication(m1, m2, m3);
	cout << "Result Matrix:" << endl;
	Matrix_Display(m3);

	Matrix_Free(m1);
	Matrix_Free(m2);
	Matrix_Free(m3);
}

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

```
	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	Matrix_Display(m1);

	float scalar = 3;

	Matrix_Multiplication(m1, scalar, m2);
	cout << "Result Matrix:" << endl;
	Matrix_Display(m2);

	Matrix_Free(m1);
	Matrix_Free(m2);
}

void TEST_DIVISION()
{
	PrintMessageInFrame("SCALAR DIVISION TEST");
	Matrix m1, m2;
	Matrix_Allocate(m1, 2, 3);
	Matrix_FillByData(m1, GetRandomData(2, 3));
	Matrix_Display(m1);
	float scalar = 3;
	Matrix_Division(m1, scalar, m2);
	cout << "Result Matrix:" << endl;
	Matrix_Display(m2);
	Matrix_Free(m1);
	Matrix_Free(m2);
}
```

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. Call `Matrix_Allocate` function for the given result `Matrix` and for required sizes and performs matrix addition for the given first two matrixes by saving the result into the `Matrix` named result.
7. Call `Matrix_Allocate` function for the given result `Matrix` and for required sizes and performs matrix substruction for the given first two matrixes by saving the result into the `Matrix` named result.
8. Call `Matrix_Allocate` function for the given result `Matrix` and for required sizes and performs matrix multiplication for the given first two matrixes by saving the result into the `Matrix` named result.
9. Call `Matrix_Allocate` function for the given result `Matrix` and for required sizes and performs scalar matrix multiplication (Multiply each element of the data member by the given constant value) for the given matrix by saving the result into the `Matrix` named result.
10. Similar to Tip (8) but for division.