# National University of Computer and Emerging Sciences


## Laboratory Manual

*for*

## Data Structures Lab

| | |
|---|---|
| Course Instructor | Dr. Amna Khan |
| Lab Instructor(s) | Asad Ullah<br>Maham Naeem |
| Section | CS-F |
| Date | 09-Sep-2020 |
| Semester | Fall 2020 |

**Department of Computer Science**

FAST-NU, Lahore, Pakistan

**Objectives:**

In this lab, students will practice:

1. Pointers
2. Templates

**Question 1**

**a.** Create a template class "Matrix" with the following members:

T** matrix;

int rows, columns

**You need to define the following member functions:**

1. An overloaded constructor which takes the values of rows and columns, and declares the required memory for the matrix. `Matrix(int rows, int columns)`

2. Copy Constructor to deep copy another matrix `Matrix(Matrix const &obj)`
   .
3. Insert function to insert an element in the given row number and column number
   `void insertElement(T const& element, int rowNo, int colNo)`

4. An overloaded + operator to add corresponding elements of two matrices. If there is a mismatch of number of rows or columns for the matrices, the operator will print an error. `Matrix<T> operator+(Matrix const& obj)`

5. A function named "print" to print the matrix in a neat and readable way. `void print()`;

6. Transpose function to take transpose of the matrix. (Convert rows into columns and vice versa). `void transpose()`

7. A destructor to delete the memory. `~Matrix()`

**b. Now test your code for the following objects in your main function:**

```
Matrix<int> m1(2, 3);
m1.insertElement(1, 0, 0);
m1.insertElement(1, 0, 1);
m1.insertElement(1, 0, 2);
m1.insertElement(0, 1, 0);
m1.insertElement(0, 1, 1);
m1.insertElement(0, 1, 2);
m1.transpose();
Matrix<int> m2(2, 3);
m2.insertElement(-1, 0, 0);
m2.insertElement(-1, 0, 1);
m2.insertElement(-1, 0, 2);
m2.insertElement(10, 1, 0);
m2.insertElement(5, 1, 1);
m2.insertElement(1, 1, 2);
m2.transpose();
Matrix<int> m3(m2);
Matrix<int> m4(m1 + m3);
```

```
m4.transpose();
m4.print();
```

**Question 2**

**a.** Write a specialized template class for char* datatype. The matrix in such a case will be like a 3-dimensional character array. Memory for each element in the matrix will be allocated dynamically.

**1.** For + operator, simple append the corresponding elements of the 2nd matrix passed as parameter with the corresponding elements of the 1st matrix.
For example: Suppose there are two matrices:
**matrix1**
Computer Electrical
Business Civil

**matrix2**
Science Engineering
Administration Engineering

**Result after applying + operator:**
ComputerScience ElectricalEngineering
BusinessAdministration CivilEngineering

**2.** The memory (number of bytes) taken by an element in the matrix will not exceed the number of characters in that element. For example the number of characters in "Computer" is 8, so 8 bytes will be allocated in memory. [As a result, the size of third dimension will be variable]. Also, there must not be any **memory leaks**.

**3.** Implement all the functions/operators for the specialized class template that you have implemented for the original class template in question 1. You will have to allocate dynamic memory for each element in the matrix and also fulfill the requirements stated in point 2.

**4.** Also implement an overloaded assignment operator in this specialized class template. **`Matrix<char*> operator=(Matrix<char*> const& obj).`**

**5.** Also implement an overloaded equality operator in this specialized class template. **`bool operator==(Matrix<char*> const& obj).`**

**b.** Now test your code by running the following instruction in your main function:

```
Matrix<int> m1(2, 3);
m1.insertElement(1, 0, 0);
m1.insertElement(1, 0, 1);
m1.insertElement(1, 0, 2);
m1.insertElement(0, 1, 0);
m1.insertElement(0, 1, 1);
m1.insertElement(0, 1, 2);
m1.transpose();

Matrix<int> m2(2, 3);
m2.insertElement(-1, 0, 0);
m2.insertElement(-1, 0, 1);
```

```
m2.insertElement(-1, 0, 2);
m2.insertElement(10, 1, 0);
m2.insertElement(5, 1, 1);
m2.insertElement(1, 1, 2);
m2.transpose();

Matrix<int> m3(m2);
Matrix<int> m4(m1 + m3);
m4.transpose();
m4.print();


Matrix<char*> mA(2,2);
mA.insertElement( (char*)"Computer", 0, 0);
mA.insertElement((char*)"Electrical", 0, 1);
mA.insertElement((char*)"Business", 1, 0);
mA.insertElement((char*)"Civil", 1, 1);

Matrix<char*> mB(2, 2);
mB.insertElement((char*)"Science", 0, 0);
mB.insertElement((char*)"Engineering", 0, 1);
mB.insertElement((char*)"Administration", 1, 0);
mB.insertElement((char*)"Engineering", 1, 1);
```

Matrix<char*> mC(2, 2);
Matrix<char*> mD(2, 2);

```
mC = mA + mB;
mD=  mA + mB;

mC.transpose();
mC.print();
cout<<mC==mD;
system("pause");
```