National University of Computer and Emerging Sciences



Lab Manual 03

Object Oriented Programming

|  |  |
| --- | --- |
| Course Instructor | Ms. Syeda Tayyaba Bukhari |
| Lab Instructor (s) | Fariha Maqbool Muhammad Usama |
| Section | BDS-2B |
| Semester | Spring 2022 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

## Objectives

After performing this lab, students shall be able to handle:

* Dynamic 2D arrays, allocation and deallocation.
* Memory Leak and Dangling Pointers

**TASK 1:**

1. Write a function **int\*\* AllocateMemory(int& rows, int& cols)** that takes size of matrix (rows and columns) from user, allocates memory for the matrix and return its pointer.
2. Write a function **void InputMatrix(int\*\* matrix, const int rows, const int cols)** which takes input the values in matrix from user(console).
3. Write a function **void DisplayMatrix(int\*\* matrix, const int& rows, const int& cols)** that displays the matrix in proper format.
4. Write a function called **maxCol** that takes as parameters a pointer to a 2D array and its dimensions. It should return the largest element in each column of the array. Since there is more than one column in 2D array, you have to return a dunamic array that contains largest of each column.

For example, if the **Sample Matrix** is

1 4 8

9 1 6

5 7 2

Your function will return array containing maximum elements of all the columns i.e.

9, 7, 8

1. Write a function **void DeallocateMemory(int\*\* matrix, const int& rows)** that deallocates all the memory.

void main()

{

int rows, cols;

**//take input from user for rows and cols**

int \*\* matrix = AllocateMemory (rows, cols);

DisplayMatrix(matrix, rows, cols);

InputMatrix(matrix, rows, cols);

int \* maxColValues = maxCol (matrix, rows, cols);

for(int i=0; i<cols; i++)

cout<< \*(maxColValues + i) <<”,”;

cout<<endl;

**//deallocate matrix**

**//delete maxColValues**

}

**TASK 2:**

# Sparse Matrix

In computer programming, a matrix can be defined with a 2-dimensional array. Any array with 'm' columns and 'n' rows represents an **m X n** matrix. There may be a situation in which a matrix contains a greater number of ZERO values than NON-ZERO values. Such matrix is known as sparse matrix.

**Example**

10 //rows

7 //cols

3 0 0 1 0 0 0

0 0 2 0 0 0 1

0 0 0 0 8 0 0

1 2 3 4 0 0 0

0 0 0 0 0 0 0

7 12 9 0 8 2 8

0 8 0 0 0 0 0

0 0 0 0 1 9 2

9 0 8 0 7 0 6

8 0 0 0 0 0 2

We can store these types of matrices using less memory, by storing only non-zero entries rather than storing all entries.

Consider the following sparse matrix:

3 0 0 1 0 0 0

0 0 9 0 0 0 0

7 12 9 0 8 2 8

You can store the above matrix as:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Row** |  | **Number of elements** | **Column index** | **value** | **Column index** | **value** |  |  |  |  |  |  |  |  |
| 0 |  | 2 | 0 | 3 | 3 | 1 |  |  |  |  |  |  |  |  |
| 1 |  | 1 | 2 | 9 |  |  |  |  |  |  |  |  |  |  |
| 2 |  | 6 | 0 | 7 | 1 | 12 | 2 | 9 | 4 | 8 | 5 | 2 | 6 | 8 |

**Figure 1**

**Note:** You are required to only allocate exact size of memory. There can be all zero entries in a row.

Use **sparse.txt** to load the data. Implement

1. void **loadMarix**(char \* filename, int \*\*& matrix, int &rows, int &col)
2. void **print**(int \*\*matrix, int rows) //print the as Figure1.
3. void **Deallocation**(int \*\*&matrix, int rows)

* Implement above three function and call them appropriately in the main.
* There should be no memory leakage.