Data Structures and Algorithms Lab

Lab 04 **Marks 10**

Instructions

- Work in this lab individually. Follow the best coding practices and include comments to explain the logic where necessary.
- You can use your books, notes, handouts, etc. but you are not allowed to borrow anything from your peer student.
- Do not use any **AI** tool for help; doing so will be considered cheating and may result in lab cancellation and possible disciplinary action.
- Test your program thoroughly with various inputs to ensure proper functionality and error handling.
- Show your work to the instructor before leaving the lab to get some or full credit.

ADT: Matrix

Write a class for **2-dimensional matrices (Matrix) of integer type**. This class should store the elements of the 2-dimensional matrix in a **one-dimensional array** of integer type created dynamically. Thus, you will have to use a **mapping mechanism (formula)** to store and retrieve the individual elements.

There will be **three private data members** of this class:

- **A pointer to integer type** (used to allocate memory dynamically).
- **An integer to store the number of rows** of the matrix.
- **An integer to store the number of columns** of the matrix.

The class should support the following **operations**, ensuring that each operation meets the specified **time complexity**:

A. Constructor

- Create a constructor for initializing a matrix of any size (**specified number of rows and columns**). The constructor should **dynamically allocate memory** for storing the elements and **initialize them to zero**, or to a specified initial value if provided as an optional parameter.
- **Time Complexity:** $O(M * N)$, where **M** is the number of rows and **N** is the number of columns, as it initializes all elements.

B. Destructor

- Implement a destructor to **free any dynamically allocated memory** resources occupied by the object and set the pointer to `nullptr`.
- **Time Complexity:** $O(1)$

C. `int get(int i, int j)`

- Retrieve the value of the element stored at the **ith row** and **jth column**. **Perform bounds checking**, and if the indices are out of bounds, **throw an exception** or **display an appropriate error message** to handle the error gracefully.
- **Time Complexity:** $O(1)$

D. `void set(int i, int j, int v)`

- Set the value at the **ith row** and **jth column** to **v**. **Perform bounds checking**, and if the indices are out of bounds, **throw an exception** or **display an appropriate error message** to handle the error gracefully.
- **Time Complexity:** $O(1)$

E. `void print(void)`

- Print the matrix on the screen in **2-D form**.
- **Time Complexity:** $O(M * N)$, where **M** is the number of rows and **N** is the number of columns, as it needs to traverse all elements.

F. `void transpose(void)`

- Take the **transpose** of the matrix. If the matrix is **not square**, modify the number of rows and columns to reflect the new dimensions.
- **Time Complexity:** $O(M * N)$, since all elements must be rearranged.

G. `void printSubMatrix(int r1, int r2, int c1, int c2)`

- Display the elements of the **sub-matrix specified by the arguments**, ensuring that the **row and column ranges are within the matrix bounds**.
- **Time Complexity:** $O((r2 - r1 + 1) * (c2 - c1 + 1))$, which corresponds to the number of elements in the sub-matrix.

H. `void makeEmpty(int n)`

- Set the elements in the **first *n* rows and *n* columns to zero**. If ***n* exceeds the matrix dimensions**, limit the operation to the available rows and columns.
- **Time Complexity:** $O(N * N)$, where ***N*** is the number of rows or columns, as the function zeroes out each element.

I. `void add(Matrix first, Matrix second)`

- Add **two matrices** and **store the result in the current object** (on which this function was called). The **dimensions of both matrices must be the same** to perform the addition. If they are not, **display an appropriate error message** and **do not perform the addition**. If the addition is successful, **update the dimensions** of the current object accordingly. To add two matrices **A** and **B** and store the result in matrix **C**, the function will be called like this: **C.add(A, B)**.
- **Time Complexity:** $O(M * N)$, where **M** is the number of rows and **N** is the number of columns, as each element in both matrices needs to be processed.

Note:

Ensure that you **handle errors gracefully** and **provide meaningful error messages** when necessary. Pay attention to **memory management**, especially when **changing matrix dimensions** during operations like **add** and **transpose**.

Demonstration:

In the **main function**, create **instances of the Matrix class** and **demonstrate the functionality** of each function. **Test edge cases**, such as matrices with **zero rows or columns**, **transpose of non-square matrices**, and **invalid index accesses**.