TASKS:

1. Implement a class Memory\_Array to manage a dynamically allocated 1D array of integers. The class should provide functionalities for insertion, deletion, and retrieval of elements. Additionally, it should adhere to the Rule of Three to ensure proper resource management.

Implement following functions:

1\_MemoryArray(); // // Constructor

2\_~MemoryArray(); // Destructor

3\_ MemoryArray(const MemoryArray& other); // Copy Constructor

4\_MemoryArray& operator=(const MemoryArray& other); Copy Assignment Operator

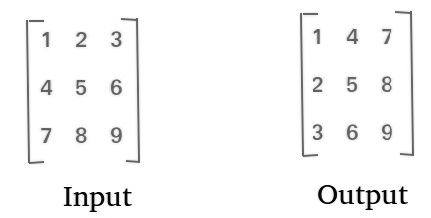
5\_void insert(int element); // Method to insert an element at the end of the array

6\_void deleteLast(); // Method to delete the last element of the arra

7\_int getElementAt(int index) const; // //Method to retrieve the element at a given index

8\_int getSize() const; //Method to get the current size of the array

1. Create a C++ program that prompts users for dimensions, allocates a **2D dynamic array**, fills it with user input, and implements a function to find the row with the highest average.



1. Implement a C++ program that uses dynamic memory allocation to create a **2D safe integer** array with dimensions’ rows and cols. Then, allocate memory for a new 2D array to store the transpose of the original array. Calculate the transpose and display both the original and transposed arrays.

Remember to handle memory allocation, user **input**, **array manipulation**, and **deallocation** (destructor for 2d) properly in your solutions. These questions will help you practice working with DMA and 2D arrays in C++.

1. Develop a C++ program to manage a jagged array representing student marks in courses. Each student has 2 to 7 courses with corresponding marks. Implement input for student data and marks, calculate the average marks for each student, and display the results. Use dynamic memory allocation for flexible course count per student.
2. Write a program that creates a **2D safe array** of **5x5** values of **type Boolean**. Suppose indices represent people and the value at **row i**, **column j** of a **2D array** is true just in case **i and j** are **friends** and **false otherwise**. Initialize your array to represent the following configuration: (**\* means “friends”**)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| I | 0 | 1 | 2 | 3 | 4 |
| 0 |  | \* |  | \* | \* |
| 1 | \* |  | \* |  | \* |
| 2 |  | \* |  |  |  |
| 3 | \* |  |  |  | \* |
| 4 | \* | \* |  | \* |  |

1. Consider a class Database Connection that establishes a connection to a remote database. This class uses a custom resource management system for connection handling. Explain the potential pitfalls if the Rule of Three is not adhered to and provide the implementations for the necessary methods.
2. Implement a C++ class for a **Book** that includes attributes for the book's **title** and **author**. Ensure proper memory management by defining **copy constructor**, **copy assignment** operator, and **destructor** to follow the "Rule of Three." Test the class by creating instances and performing assignments, demonstrating correct memory handling.

Implement following class?

class Book {

private:

char\* title\_;

char\* author\_;

public:

Book (const char\* title, const char\* author);

Book (const Book& other);

Book& operator= (const Book& other);

~Book (); // ... other methods and attributes ...

};

In main test using:

Book book1("The Catcher in the Rye", "J.D. Salinger");

Book book2 = book1; // Test copy constructor

Book book3("To Kill a Mockingbird", "Harper Lee");

book3 = book2;