Matrix Operations in C++

Description:

This code demonstrates various matrix operations in C++. It includes functions to print a matrix in row-wise and column-wise representations, perform linear search on a matrix, and input a matrix from either user-defined dimensions or a predefined matrix. Additionally, it showcases the usage of these functions in the **main** function.

Matrix in C++:

A matrix is a two-dimensional data structure that represents a collection of elements arranged in rows and columns. In C++, a matrix can be represented using a 2D array. Each element in the matrix can be accessed using its row and column indices.

Functionality and Complexity:

1. **printRowWiseMatrix**:
    - Description: Prints the matrix in row-wise representation.
    - **Time Complexity: O(ROWS * COLS) - traversing all elements of the matrix.**
    - **Space Complexity: O(1) - no additional memory is used.**
2. **printColumnMatrix**:
    - Description: Prints the matrix in column-wise representation.
    - **Time Complexity: O(ROWS * COLS) - traversing all elements of the matrix.**
    - **Space Complexity: O(1) - no additional memory is used.**
3. **linearSearch**:
    - Description: Performs linear search on a matrix and returns the position of the target element.
    - **Time Complexity: O(ROWS * COLS) - traversing all elements of the matrix in the worst case.**
    - **Space Complexity: O(1) - no additional memory is used.**
4. **matrix**:
    - Description: Inputs a matrix from the user and prints it in row-wise and column-wise representations.
    - **Time Complexity: O(ROWS * COLS) - inputting and printing all elements of the matrix.**
    - **Space Complexity: O(ROWS * COLS) - storing the matrix elements in a 2D array.**

5. **userInputMatrix**:
    - Description: Inputs a matrix of user-defined size from the user and prints it in row-wise and column-wise representations.
    - **Time Complexity: O(row * col) - inputting and printing all elements of the matrix.**
    - **Space Complexity: O(row * col) - storing the matrix elements in a 2D array.**

Memory Allocation:

The matrices in this code are declared as 2D arrays with a fixed size using the **#define** directive. The space for these matrices is allocated statically at compile-time. For user-defined matrices, memory is allocated dynamically based on the user's input for the row and column dimensions.

To find the address of a specific element in a 2D array (matrix), you can use the following formula:

```
address = baseAddress + (row * numColumns + column) * sizeof(elementType)
```

Where:

- **address** is the calculated address of the element.
- **baseAddress** is the memory address of the first element of the matrix.
- **row** is the row index of the desired element (starting from 0).
- **numColumns** is the number of columns in the matrix.
- **column** is the column index of the desired element (starting from 0).
- **sizeof(elementType)** is the size in bytes of each element in the matrix.