2D Array Practice Questions

📘 Theoretical Questions

1. In a 2D array, what is the primary difference between row-major and column-major memory layouts?
   a) Row-major stores elements sequentially by rows, while column-major stores them sequentially by columns.
   b) Row-major stores elements sequentially by columns, while column-major stores them sequentially by rows.
   c) Row-major uses dynamic allocation, while column-major uses static allocation.
   d) There is no difference.

2. The formula "Address = Base + s × (i × number_of_columns + j)" is used to compute the address of an element in a 2D array using which indexing scheme?
   a) Column-major order
   b) Row-major order
   c) Diagonal order
   d) Zigzag order

3. For a 2D array declared as `int arr[3][4]` in C, where each element occupies 4 bytes, what is the total memory occupied by the array?
   a) 12 bytes
   b) 48 bytes
   c) 24 bytes
   d) 16 bytes

4. When initializing a 2D array at compile time, which of the following is true?
   a) The values must be provided explicitly in the source code.
   b) The values are determined during run-time via user input.
   c) The array is always dynamically allocated on the heap.
   d) Only the first row can be initialized at compile time.

5. In pointer arithmetic with 2D arrays in C, the expression `*(*(arr + i) + j)` returns:
   a) The address of the element at row `i` and column `j`.
   b) The value of the element at row `i` and column `j`.
   c) The number of elements in the array.
   d) The size of row `i`.

6. How must the memory address formula be adjusted for 1-based indexing (instead of 0-based) in a row-major 2D array?
   a) Subtract 1 from both row and column indexes in the formula.
   b) Add 1 to the base address.
   c) Multiply the total offset by 1.
   d) No adjustment is needed.

7. Which method is commonly used to dynamically allocate a 2D array in C?
   a) Allocating a contiguous block of memory and simulating 2D indexing.
   b) Using two-dimensional stack allocation only.
   c) Declaring a fixed-size array with preset dimensions.
   d) Initializing with compile-time constants.

8. Which of the following languages typically uses a row-major order for multi-dimensional arrays?
   a) Fortran
   b) C and C++
   c) MATLAB
   d) None of the above

9. For a symmetric square matrix, which relationship must hold true for all elements?
   a) Every element is greater than its adjacent elements.
   b) The element at row `i`, column `j` is equal to the element at row `j`, column `i`.
   c) The sum of each row is equal to the sum of each column.
   d) Elements are arranged in increasing order row-wise.

10. Which of the following is NOT a direct benefit of understanding the memory layout of 2D arrays?
    a) Performance optimization through better cache utilization.
    b) Simplifying pointer arithmetic when accessing elements.
    c) Enhancing algorithm efficiency by choosing appropriate traversal methods.
    d) Automatically preventing segmentation faults.

💻 Coding Questions

11. **C++ – Transpose a Matrix**
    Write a C++ program to input a 2D array (matrix), compute its transpose, and display both the original and transposed matrices.

```cpp
#include <iostream>
using namespace std;
int main() {
    int rows, cols;
    cin >> rows >> cols;
    int arr[100][100], trans[100][100];
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            cin >> arr[i][j];
    // Compute transpose
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < cols; j++)
            trans[j][i] = arr[i][j];
    // Display original matrix
    cout << "Original Matrix:" << endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++)
            cout << arr[i][j] << " ";
        cout << endl;
    }
    // Display transposed matrix
    cout << "Transposed Matrix:" << endl;
    for (int i = 0; i < cols; i++) {
        for (int j = 0; j < rows; j++)
            cout << trans[i][j] << " ";
        cout << endl;
    }
    return 0;
}
```

12. **Python – Check Anti-Symmetric Matrix**

Implement a Python function to check if a square matrix is anti-symmetric (i.e. `A^T = -A`).

```python
def is_antisymmetric(matrix):
    n = len(matrix)
    for i in range(n):
        for j in range(n):
```

```
            if matrix[i][j] != -matrix[j][i]:
                return False
    return True

# Example usage:
mat = [
    [0,  2, -1],
    [-2, 0, -4],
    [1,  4,  0]
]
print(is_antisymmetric(mat))
```

13. **Java – Dynamic Allocation and Boundary Elements**

Write a Java program that dynamically allocates a 2D array, fills it with numbers, and prints its boundary elements in clockwise order.

java

```
import java.util.Scanner;
public class BoundaryElements {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int rows = sc.nextInt();
        int cols = sc.nextInt();
        int[][] arr = new int[rows][cols];
        // Input elements
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                arr[i][j] = sc.nextInt();
        // Print boundary in clockwise order
        // Top row
        for (int j = 0; j < cols; j++)
            System.out.print(arr[0][j] + " ");
        // Right column (excluding top)
        for (int i = 1; i < rows; i++)
            System.out.print(arr[i][cols - 1] + " ");
        // Bottom row (if not the same as top, in reverse)
        if (rows > 1) {
            for (int j = cols - 2; j >= 0; j--)
                System.out.print(arr[rows - 1][j] + " ");
        }
        // Left column (if not the same as right, in reverse)
```

```
        if (cols > 1) {
            for (int i = rows - 2; i > 0; i--)
                System.out.print(arr[i][0] + " ");
        }
        sc.close();
    }
}
```

14. **C – Run-Time Initialization and Display**

    Write a C program to initialize a 2D array at run time (using user input) and then print the array.

```c
#include <stdio.h>
int main() {
    int m, n;
    scanf("%d%d", &m, &n);
    int arr[m][n];
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &arr[i][j]);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++)
            printf("%d ", arr[i][j]);
        printf("\n");
    }
    return 0;
}
```

15. **Python – Rotate Matrix 90° Counter-Clockwise**

    Write a Python function to rotate a 2D square matrix 90 degrees counter-clockwise.

```python
def rotate_counterclockwise(matrix):
    n = len(matrix)
    # Transpose the matrix
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
    # Reverse each column
```

```
    for j in range(n):
        for i in range(n // 2):
            matrix[i][j], matrix[n - i - 1][j] = matrix[n - i - 1][j], matrix[i]
[j]
    return matrix

# Example usage:
mat = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
rotated = rotate_counterclockwise(mat)
for row in rotated:
    print(row)
```

16. **C++ – Sum of Rows and Columns**

Write a C++ program that calculates and displays the sum of the elements in each row and each column of a 2D array.

```cpp

#include <iostream>
using namespace std;
int main() {
    int m, n;
    cin >> m >> n;
    int arr[100][100];
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> arr[i][j];
    // Sum of rows
    for (int i = 0; i < m; i++) {
        int rowSum = 0;
        for (int j = 0; j < n; j++)
            rowSum += arr[i][j];
        cout << "Sum of row " << i << ": " << rowSum << endl;
    }
    // Sum of columns
    for (int j = 0; j < n; j++) {
        int colSum = 0;
        for (int i = 0; i < m; i++)
```

```
            colSum += arr[i][j];
        cout << "Sum of column " << j << ": " << colSum << endl;
    }
    return 0;
}
```

17. **Java – Row with Maximum Sum**

Write a Java method that takes a 2D array and returns the index of the row with the maximum sum.

```java
public class MaxRowSum {
    public static int maxRowIndex(int[][] arr) {
        int maxSum = Integer.MIN_VALUE, maxIndex = -1;
        for (int i = 0; i < arr.length; i++) {
            int sum = 0;
            for (int j = 0; j < arr[i].length; j++)
                sum += arr[i][j];
            if (sum > maxSum) {
                maxSum = sum;
                maxIndex = i;
            }
        }
        return maxIndex;
    }
    public static void main(String[] args) {
        int[][] matrix = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
        System.out.println("Row with maximum sum: " + maxRowIndex(matrix));
    }
}
```

18. **C – Find Saddle Point**

Write a C program to find the saddle point in a 2D array (an element which is the minimum in its row and maximum in its column). Assume the array is of fixed size.

```c
#include <stdio.h>
#include <limits.h>
int main() {
    int arr[3][3], i, j, saddle = INT_MIN, colIndex;
```

```
    for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
            scanf("%d", &arr[i][j]);
    for(i = 0; i < 3; i++){
        int minRow = arr[i][0];
        colIndex = 0;
        for(j = 1; j < 3; j++){
            if(arr[i][j] < minRow){
                minRow = arr[i][j];
                colIndex = j;
            }
        }
        int k;
        int isSaddle = 1;
        for(k = 0; k < 3; k++){
            if(arr[k][colIndex] > minRow){
                isSaddle = 0;
                break;
            }
        }
        if(isSaddle){
            saddle = minRow;
            printf("Saddle point is: %d\n", saddle);
            return 0;
        }
    }
    printf("No saddle point found\n");
    return 0;
}
```

19. **Python – Flatten a 2D Array**

Write a Python function that flattens a 2D array into a 1D list (row-wise).

python

```python
def flatten(matrix):
    return [element for row in matrix for element in row]

# Example usage:
mat = [
    [1, 2],
    [3, 4],
```

```
    [5, 6]
]
print(flatten(mat))
```

20. Java – Matrix Multiplication

Write a Java program to multiply two 2D arrays (matrices) and print the resulting matrix.

```java
public class MatrixMultiplication {
    public static void main(String[] args) {
        int[][] A = {
            {1, 2},
            {3, 4}
        };
        int[][] B = {
            {5, 6},
            {7, 8}
        };
        int rowsA = A.length, colsA = A[0].length;
        int rowsB = B.length, colsB = B[0].length;
        if (colsA != rowsB) {
            System.out.println("Matrix multiplication not possible.");
            return;
        }
        int[][] result = new int[rowsA][colsB];
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++) {
                for (int k = 0; k < colsA; k++) {
                    result[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        // Print the resulting matrix
        for (int i = 0; i < rowsA; i++) {
            for (int j = 0; j < colsB; j++)
                System.out.print(result[i][j] + " ");
            System.out.println();
        }
    }
}
```

✅ Answer Key (Theoretical Questions)

Q#	Answer
1	a
2	b
3	b
4	a
5	b
6	a
7	a
8	b
9	b
10	d

2/2