Multi-dimensional Arrays & Singly Linked Lists

1. Rotate a 2D matrix (image) by 90 degrees clockwise without using extra space.
2. Design a system to track parked cars in a 3-level parking lot (3D array). Show availability.
3. Create a playlist where each song points to the next. Add necessary functionality
4. Simulate a train with compartments dynamically added/removed.
5. Reverse the order of words in a sentence using a linked list

Weeks 3 & 4: Doubly/Circular/Sorted Linked Lists

6. Implement back/forward navigation for a browser.
7. Allow undo/redo actions but discard oldest actions after 10 steps
8. Loop a playlist infinitely (e.g., "radio mode").
9. Maintain a list of employees sorted by ID.
10. Store real-time sensor readings with overwrite on overflow.

Weeks 5 & 6: Stacks

11. Validate nested brackets in code (e.g., `[{()}]`).
12. Verify if a string reads the same backward (e.g., "madam").
13. Convert infix to postfix notation.
14. Allow users to close the most recently opened tab first.

Weeks 7–9: Queues & Sorting Algorithms

15. Manage print requests in FIFO order with cancellation.
16. Sort products by price using selection sort.
17. Check if two strings are anagrams using sorting.
18. Sort student records by roll number
19. Sort votes in descending order

Weeks 10 & 11: Searching & Recursion

20. Find if a word exists in a grid (DFS with recursion).
21. Find the two nearest points in a 2D plane.
22. Guess a number between 1–1000 using binary search.
23. Compute the nth Fibonacci number using recursion
24. Print all permutations of a string.

Weeks 12 & 13: Trees & Traversals

25. Compute the value of `(2+3)*4` using a tree using Post-order traversal.
26. Find the lowest common ancestor of two people. Hint: Parent pointer
27. Compare two trees for differences.
28. **Threaded In-Order Traversal**: Optimize in-order traversal without a stack.
29. **Huffman Coding Compressor**: Build a Huffman tree for text compression.

Weeks 14 & 15: Heaps & Graphs, Hashing

30. Prioritize patients using a max-heap.
31. Find mutual friends using adjacency lists using Graph traversal (BFS/DFS).
32. Navigate between cities using Dijkstra's algorithm.
33. Connect all cities with minimal road cost.
34.

1. Library Shelf Tracker A library uses a 2D grid (3 rows x 3 columns) to organize books. Each shelf slot can hold one book title.

- Task: Write a C++ program to:
    - Input 9 book titles into the grid.
    - Allow the user to search for a book by row and column (e.g., input `row=1`, `col=2` should return the 2nd row, 3rd column).
    - Print the entire grid.

- Sample Input: Row 0, Col 0: "Harry Potter" Row 0, Col 1: "Lord of the Rings" ... (9 titles total)	Sample Output: [Row 0] ["Harry Potter", "Lord of the Rings", .] [Row 1] [D, E, F] [Row 2] [G, H, I] Book at (1,2): F

2. Train Compartment Management A train has 3 compartments: A → B → C. Each compartment has a name and a pointer to the next compartment.

- Task:
    - Create a singly linked list to model the train.
    - Insert a new compartment "D" between B and C.
    - Traverse and print the updated compartments.

3. Undo Feature for a Text Editor : A text editor allows users to undo their last 3 actions (e.g., "Type Hello", "Delete Word", "Bold Text").

- Task:
    - Use an array-based stack to store the last 3 actions.
    - Implement `push(action)` and `pop()` to undo the most recent action.
    - Print the remaining actions after each undo.

4. Browser History with Linked List Stack A browser stores the last 5 URLs visited. The "Back" button removes the most recent URL from the history.

- Task:
    - Use a linked list-based stack to manage URLs.
    - After visiting 5 URLs (e.g., "google.com", "youtube.com", ...), simulate clicking "Back" twice.
    - Print the remaining history.

5. Ticket Counter Simulation

- A ticket counter serves customers in FIFO order. The queue can hold up to 10 people.
- Task:
    - Use a circular queue (array of size 10) to enqueue 12 people (simulate overflow).
    - Dequeue the first 3 people.
    - Print the queue after each operation.

6. Printer Job Management

-A printer processes documents in the order they are received.

- Task:
    - Use a linked list-based queue to add 5 documents ("Doc1" to "Doc5").
    - Process (dequeue) the first document.
    - Print the remaining queue.
- Sample Output:

Processing Doc1. Remaining: Doc2 → Doc3 → Doc4 → Doc5

7. Bubble Sort Visualization

-: A teacher wants to show students how bubble sort works step-by-step.

- Task:
    - Sort the array `[5, 2, 8, 1]` in ascending order.
    - Print the array after each pass of bubble sort.
- Sample Output:

    Pass 1: [2, 5, 1, 8]
    Pass 2: [2, 1, 5, 8]
    Pass 3: [1, 2, 5, 8]

8. Inventory Search

- A shopkeeper’s inventory is stored as `["pen", "book", "pencil"]`.
- Task:
    - Write a function `int linearSearch(string item)` that returns the index of "pencil".
    - Handle the case where the item is not found.
- Sample Output:

    "pencil" found at index 2.

1. In-Place Matrix Rotation

-: Rotate a 4x4 matrix 90° clockwise without using extra memory.

- Task:
    - Implement the rotation by transposing the matrix and reversing rows.
    - Test with input:

    Input Matrix:
    1 2 3 4
    5 6 7 8
    9 10 11 12
    13 14 15 16

    - Print the rotated matrix.
- Sample Output:

    13 9 5 1
    14 10 6 2
    15 11 7 3
    16 12 8 4

2. Sorted Doubly Linked List Insertion

-: A sorted DLL contains nodes with values `[10, 20, 30]`. Insert a new node with value 25 while maintaining order.

- Task:
    - Implement insertion in O(n) time.
    - Handle edge cases (e.g., inserting at head/tail).
    - Print the DLL before and after insertion.
- Sample Output:

    Before: 10 ↔ 20 ↔ 30
    After: 10 ↔ 20 ↔ 25 ↔ 30

3. Min-Stack with O(1) Operations

- Design a stack that supports `push`, `pop`, and `getMin` in constant time.
- Task:
    - Use two stacks: one for main data and one for tracking minima.
    - Test with operations: `push(3)`, `push(5)`, `push(2)`, `getMin()`, `pop()`, `getMin()`.
    - Print the minimum after each operation.
- Sample Output:

    After push(3): Min = 3
    After push(5): Min = 3
    After push(2): Min = 2
    After pop(): Min = 3

4. Balanced Brackets Checker

-: Validate if a string like `"{[()]}"` has properly nested brackets.

- Task:
    - Use a stack to track opening brackets.
    - Return `true` for valid strings, `false` otherwise.
    - Test cases: `"([)]"` (invalid), `"()[]{}"` (valid).
- Sample Output:

    "([)]" is invalid.
    "()[]{}" is valid.