**15/04/2025**

**EDSA Assignment Report**

Prathul Dev CP

ME24I1003

Question 3: Autonomous Crane Fabrication Unit Simulation using Data Structures

**1. What Problem Are You Solving?**

This simulation models the operational logic of an **Autonomous Crane Fabrication Unit**, focusing on how different data structures manage components through the stages of **assembly**, **storage**, **fault tracking**, **repairing**, and **prioritization**. It mimics real-world logistics and maintenance workflows using classical DSA concepts.

**2. Key Objectives**

* Simulate a real-life crane factory’s component and machine handling using DSA.
* Efficiently manage cranes and their parts through various operational states.
* Demonstrate use of **queue**, **stack**, **array**, **singly/doubly/circular linked lists**.
* Enable easy visualization of data flow and state transitions in a factory setting.

**3. Design Explanation**

**Why These Data Structures?**

* **Queue (FIFO):** For loading components in trucks in their original sequence.
* **Stack (LIFO):** For unloading components in reverse order — simulates truck unloading.
* **Array with Overflow Handling:** For compact crane storage with overwrite logic when full.
* **Singly Linked List:** To track faulty cranes — dynamic and easy insertions/removals.
* **Doubly Linked List:** For managing repaired cranes — allows forward and backward inspection.
* **Circular Linked List:** For looping over priority cranes — supports endless upgrades.

**Efficiency Rationale**

Each data structure aligns with an actual operation:

* Stack and Queue handle transport order naturally.
* Arrays with circular overflow prevent memory overuse while enabling overwriting.
* Linked lists provide flexibility for dynamic, non-contiguous memory allocation.

**4. Logic of the Code (Step-by-Step)**

**Part A: Assembly Process**

1. Crane components (Boom, Hook, etc.) are enqueued.
2. Components are then dequeued and pushed onto a stack.
3. Stack is popped (LIFO) to simulate reverse unloading from truck.

**Part B: Storage Unit**

1. Cranes are stored in a fixed-size array.
2. After reaching max capacity, older cranes are overwritten cyclically.
3. Mimics limited lobby storage with replacement logic.

**Part C: Fault Tracking**

1. Faulty cranes are added using a **singly linked list**.
2. Once repaired, they are moved into a **doubly linked list**.
3. Supports insert, delete, and bidirectional traversal.

**Part D: Priority Upgrades**

1. Cranes needing urgent upgrades are added to a **circular linked list**.
2. List is traversed multiple rounds for repetitive inspections or updates.

**5. Variables and Functions Used**

***Important Variables:***

* queue[SIZE][LENGTH]: Stores crane components in FIFO manner.
* stack[SIZE][LENGTH]: Temporary holding of components in LIFO fashion.
* storage[SIZE][LENGTH]: Crane storage with overwrite logic.
* nextCrane: Circular index for storage overwrite.
* struct node \*: For linked lists (faulty, repaired, and priority cranes).

***Key Functions:***

* **enqueue() / dequeue()**: Queue logic for part assembly.
* **push() / pop()**: Stack logic for reverse loading.
* **storeCrane()**: Handles crane storage with overwrite once full.
* **insertFaulty() / deleteFaulty()**: Singly linked list insert/delete.
* **insertRepaired() / traverseFwd() / traverseBack()**: Doubly linked list operations.
* **insertPriority() / transversePriority()**: Circular list for repeating crane upgrades.