**Elementary Data Structure and Logical Thinking**

**Report**

Assignment Question 2: Deep-Sea Rover Repair Station

Name: S Shri Rishabh

Roll Number: ME24B1012

Branch: B.Tech Mechanical

Subject: Elementary Data Structure and Logical Thinking

Language Used: C Programming

Submission Date: 21.04.2025

Objective:

The objective of this assignment is to apply essential data structures such as queues, stacks, arrays, and linked lists to simulate the operations of a Deep-Sea Rover Repair Station. The station is tasked with maintaining rover parts and managing their repair, assembly, and upgrades for ocean floor exploration.

This simulation demonstrates how mechanical engineering problems, such as part flow, storage, maintenance, and upgrades, can be efficiently managed through computer science principles.

Scenario Summary:

In this scenario, we are designing a Deep-Sea Rover Repair Station that needs to handle operations such as:

Part Delivery System: Parts arrive for maintenance and are queued for processing.

Arm Task Manager: Parts are stacked in a LIFO (Last In, First Out) order to prioritize the most recently needed parts.

Assembly Storage Unit: Repaired rovers are stored in a fixed-size array. Once full, the oldest rover is deployed.

Repair and Upgrade Tracker: Faulty rovers are tracked using a singly linked list, repaired rovers are stored in a doubly linked list, and high-priority rovers are managed in a circular linked list for urgent upgrades.

Approach and Logic Explained:

Part Delivery and Arm Task Manager – Queue and Stack

Logic: Parts like "Drill", "Wheel", "Camera", etc., arrive sequentially through a submersible, and we use a Queue (FIFO) to store them. However, certain parts may need to be stacked in reverse order (using a Stack (LIFO)) to prioritize the latest delivery for processing.

Implementation Steps:

Enqueue parts into the queue.

Dequeue parts and push onto a stack.

Pop from the stack to simulate the repair order.

Why it works: The LIFO structure ensures that the most recent parts (such as "Sonar" for navigation readiness) are handled first to ensure the rover is fully equipped for its mission.

Assembly Storage Unit – Fixed Size Array

Logic: Repaired rovers are stored in an array with a fixed size of 5 slots. Once the array is full, the oldest rover is deployed to make space for new rovers.

Implementation Steps:

Insert the first 5 rovers into the array.

When the array is full, deploy the oldest rover to make room for the newest one.

Why it works: This approach mimics the management of limited storage space, ensuring that the most recent rovers are readily available, while older rovers are dispatched to make room for more urgent repairs or upgrades.

Faulty Rover Tracker – Singly and Doubly Linked Lists

Logic: Faulty rovers (such as "Rov3" and "Rov6") are initially added to a singly linked list. Once repaired, they are moved to a doubly linked list for easier inspection and maintenance.

Implementation Steps:

Insert faulty rovers into the singly linked list.

After repair, delete from the singly linked list and insert into the doubly linked list.

Traverse the doubly linked list both forwards and backwards for easy inspection.

Why it works: The singly linked list is efficient for tracking faulty rovers, while the doubly linked list allows bidirectional inspection of repaired units, making the maintenance process more efficient.

Priority Upgrades – Circular Linked List

Logic: Rovers requiring urgent upgrades (such as "Rov1" and "Rov4") are added to a circular linked list, which allows continuous looping through the list to ensure that these high-priority units are not missed.

Implementation Steps:

Insert "Rov1" and "Rov4" into the circular linked list.

Traverse the list twice to simulate the cycling process.

Why it works: The circular nature of the linked list ensures that urgent upgrades can be revisited continuously until completed, simulating the urgent nature of critical upgrades in the repair process.

Task Breakdown:

a) Part Delivery and Arm Task Manager

Enqueue the parts into a queue (e.g., "Drill", "Wheel", "Camera", "Arm", "Battery", "Sonar").

Dequeue them one by one and push onto a stack.

Pop the stack to show the reverse repair order, starting with the most recent part ("Sonar").

Creativity Bonus:

Using LIFO for the "Sonar" part ensures that this critical component is repaired last, ensuring the rover's navigation system is upgraded first for optimal performance.

b) Assembly Storage Unit

Insert 7 rovers into a 5-slot array.

When the array is full, remove the oldest rover to make space for the new ones ("Rov1", "Rov2", ..., "Rov7").

Creativity Bonus:

The deployment of the oldest rover helps mitigate corrosion risk from prolonged exposure to saltwater, ensuring the newest rovers are always in top condition.

c) Faulty Rover Tracker

Insert "Rov3" and "Rov6" into a singly linked list for repairs.

Once repaired, move "Rov3" to the doubly linked list for further inspection and maintenance.

Creativity Bonus:

"Rov3's" wheel was jammed by sand particles and was cleaned using pressure jets, which were able to clear the obstruction efficiently.

d) Priority Upgrades

Insert "Rov1" and "Rov4" into a circular linked list for urgent upgrades.

Traverse the list twice to simulate the cycling through high-priority upgrades.

Creativity Bonus:

"Rov4" receives a bioluminescent shell upgrade, making it more visible for deep-sea operations and ensuring it can easily be tracked by the repair station.

Overall Learning and Conclusion:

This assignment provided an opportunity to integrate mechanical engineering with computer science by applying data structures to simulate the operations of a Deep-Sea Rover Repair Station. The exercise enhanced my understanding of how basic data structures such as queues, stacks, arrays, and linked lists can be used to model real-world scenarios, improving both engineering processes and system management. It also reinforced the importance of selecting appropriate data structures for different tasks, leading to more efficient and scalable solutions.

Through this task, I was able to deepen my programming skills in C and gain practical insights into how software can assist in managing complex mechanical systems and logistics operations.

Sample output:



