ELEMENTARY DATA STRUCTURES AND LOGICAL THINKING

ASSIGNMENT

Name- R. Lakshay Vardhan

Roll number- ME24B1001

Date- 15/04/2025

**OBJECTIVES-**

The problem which I solved using the elementary data structures like stack, queues, arrays, linked lists (singly, doubly, circular) is simulation of futuristic robotic car assembly line.

The above data structures have been implemented to perform following operations-

* Managing part deliveries using a **Queue**.
* Simulating robotic assembly using a **Stack**.
* Storing assembled cars using a fixed-capacity **Array**.
* Tracking defective and repaired cars using **Linked Lists** (Singly, Doubly, and Circular).

Each part of the system mimics a real-world mechanical process in a robotic assembly plant, showcasing how data structures streamline task handling.

**DESIGN AND RATIONALE FOR DATA STRUCTURES-**

**Queue**- Used for the Part Delivery System to represent a first-come-first-serve approach where car parts arrive in order.

* Rationale: FIFO nature of a queue matches the conveyor belt delivery model, since the part which enters the belt is the one which is removed the first.

**Stack**- Used for the Robot Arm Task Manager to assemble parts in LIFO order.

* Rationale: The last part added (e.g., the hood) needs to be on top for easy access and finishing, so LIFO makes sense.

**Array (Fixed-size)** - Used for Garage Storage of prototypes.

* Rationale: A fixed-size array (like an actual garage with limited slots) supports predictable access and removal of the oldest car for space.

**Singly Linked List**- For Defective Cars Tracker.

* Rationale: Simple insertion and traversal to track defective cars.

**Doubly Linked List**- For Repaired Cars Tracker.

* Rationale: Allows forward and backward movement through repaired items for inspections.

**Circular Linked List** - For VIP Car Upgrades.

* Rationale: Allows continuous cycling through VIP cars for ongoing checks.

**LOGIC EXPLANATION-**

**Part A: Delivery and Assembly**

* Parts like "Engine", "Chassis", etc., are enqueued using a queue.
* Robot arm dequeues each part and pushes them onto a stack.
* During dequeue process, the first enqueued element gets dequeued (FIFO property).
* Parts are popped from the stack to simulate the reverse assembly order.
* During pop, element which is pushed recently gets popped (LIFO property).

**Part B: Garage Storage**

* The garage has 8 slots (array size).
* On adding more than 8 cars (e.g., Car9, Car10), the oldest (first inserted) is removed, simulating shipment.

**Part C: Defective and Repaired Cars**

* "Car3" and "Car6" are added to a singly linked list.
* "Car3" is later removed and added to a doubly linked list to allow detailed inspection.
* Traversal in both directions is done to validate the list.

**Part D: VIP Upgrades**

* "Car1" and "Car5" are added to a circular linked list.
* Two cycles of traversal simulate ongoing upgrade checks.

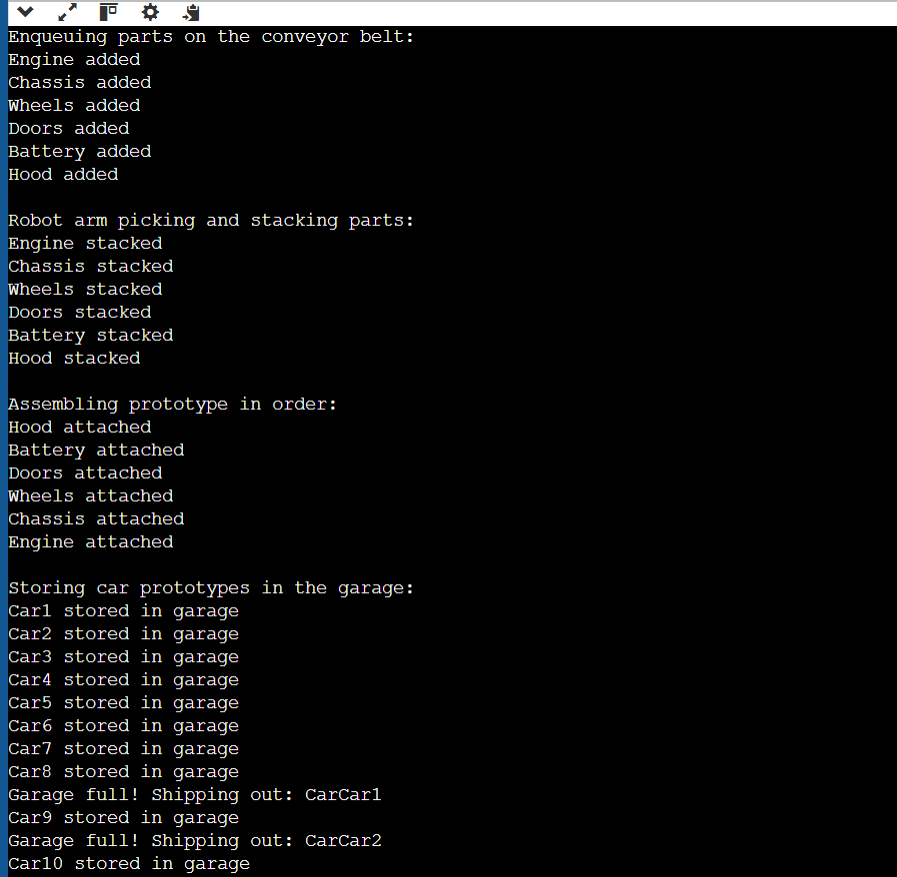
**Purpose of Variables and Functions**

**Variables:**

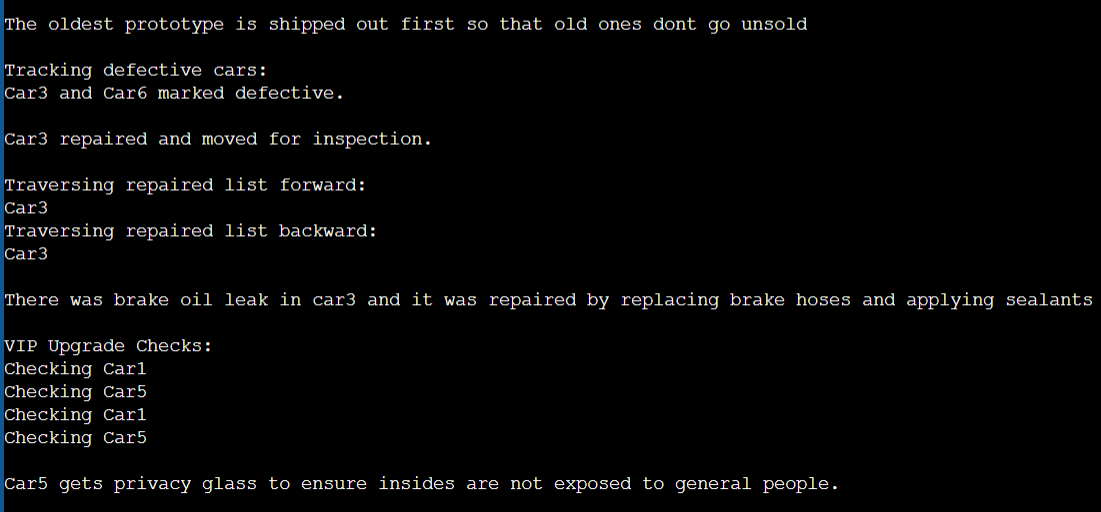
* queue[MAX\_PARTS], stack[MAX\_PARTS]: Arrays for part delivery and stack operations.
* garage[MAX\_PROTOTYPES]: Fixed-size array representing garage slots.
* front, rear, top: Index trackers for queue and stack.
* garageStart, garageEnd: Indices to manage circular garage insertion/removal.

**Functions:**

* enqueue() / dequeue(): Add/remove parts in queue.
* push() / pop(): Stack operations for robot arm.
* addToGarage(): Inserts prototypes and manages overflow in garage.
* simulate...(): Wrapper functions for each part of the simulation to run and print results.



Sample output of the code



Sample output of the code

**CONCLUSION-**

This simulation successfully uses foundational data structures to represent real-world manufacturing scenarios in a robotic assembly plant. The combination of queue, stack, arrays, and linked lists illustrates how different data access patterns support diverse automation tasks. This hands-on approach builds not only coding skills but also strengthens problem-solving and design thinking in systems engineering.