Robotic Assembly Line Simulator

Name: RAJ ROSHAN TIWARI

Roll Number: ME24B1011

Date: 15/04/2025

# Problem Statement

Design and simulate a Robotic Assembly Line for a futuristic car manufacturing plant using various data structures to model subsystems such as part delivery, robot task management, storage, repair tracking, and VIP upgrades.

# Objectives

- Simulate car part delivery and assembly using Queue and Stack.

- Store assembled prototypes in an Array with overflow handling.

- Track defective and repaired cars using Singly and Doubly Linked Lists.

- Manage VIP upgrade cars using a Circular Linked List.

# Design Explanation

Queues efficiently handle first-come-first-serve part delivery. The Stack represents the LIFO nature of robotic assembly. An Array is used for fixed-size storage. Linked Lists provide flexible tracking of prototypes in various repair stages, while the Circular Linked List allows continuous checking for VIP upgrades.

# Code Logic and Implementation

## Combined C Code Implementation

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
  
#define SIZE 6  
#define CAPACITY 8  
  
// Queue and Stack: Part Delivery and Robot Arm  
void simulate\_assembly\_line() {  
 char \*queue[SIZE] = {"Engine", "Chassis", "Wheels", "Doors", "Battery", "Hood"};  
 char \*stack[SIZE];  
 int top = -1;  
  
 for (int i = 0; i < SIZE; i++) {  
 stack[++top] = queue[i]; // Pushing into stack from queue  
 }  
  
 printf("=== Assembly Order ===\n");  
 while (top >= 0) {  
 printf("Assembling: %s\n", stack[top--]);  
 }  
 printf("\n");  
}  
  
// Array: Assembly Storage Unit  
void simulate\_garage\_storage() {  
 char garage[CAPACITY][10];  
 int count = 0;  
  
 for (int i = 1; i <= 10; i++) {  
 if (count == CAPACITY) {  
 for (int j = 1; j < CAPACITY; j++) {  
 strcpy(garage[j - 1], garage[j]);  
 }  
 count--;  
 }  
 sprintf(garage[count++], "Car%d", i);  
 }  
  
 printf("=== Garage Storage ===\n");  
 for (int i = 0; i < count; i++) {  
 printf("%s ", garage[i]);  
 }  
 printf("\n\n");  
}  
  
// Singly and Doubly Linked List: Defective Prototype Tracker  
typedef struct Node {  
 char data[10];  
 struct Node \*next;  
} Node;  
  
typedef struct DNode {  
 char data[10];  
 struct DNode \*prev, \*next;  
} DNode;  
  
void simulate\_defective\_tracker() {  
 // Singly Linked List for defective cars  
 Node \*head = malloc(sizeof(Node));  
 strcpy(head->data, "Car3");  
 head->next = malloc(sizeof(Node));  
 strcpy(head->next->data, "Car6");  
 head->next->next = NULL;  
  
 // Doubly Linked List for repaired cars  
 DNode \*car3 = malloc(sizeof(DNode));  
 strcpy(car3->data, "Car3");  
 car3->prev = NULL;  
 car3->next = NULL;  
  
 printf("=== Defective Tracker ===\n");  
 printf("Defective Cars: %s -> %s\n", head->data, head->next->data);  
 printf("Repaired Car (Forward): %s\n", car3->data);  
 printf("Repaired Car (Backward): %s\n\n", car3->data);  
}  
  
// Circular Linked List: VIP Priority Upgrades  
void simulate\_vip\_upgrade() {  
 Node \*car1 = malloc(sizeof(Node));  
 Node \*car5 = malloc(sizeof(Node));  
  
 strcpy(car1->data, "Car1");  
 strcpy(car5->data, "Car5");  
  
 car1->next = car5;  
 car5->next = car1; // Circular link  
  
 Node \*current = car1;  
  
 printf("=== VIP Upgrade Checks ===\n");  
 for (int i = 0; i < 4; i++) {  
 printf("Upgrade Check: %s\n", current->data);  
 current = current->next;  
 }  
 printf("\n");  
}  
  
// Main Function  
int main() {  
 simulate\_assembly\_line();  
 simulate\_garage\_storage();  
 simulate\_defective\_tracker();  
 simulate\_vip\_upgrade();  
 return 0;  
}

## Creativity Bonuses

- The robot arm places the 'Hood' last to ensure aesthetic finishing, enhancing the car's front design.

- Oldest prototypes are shipped out to prevent rusting and to meet delivery schedules.

- Car3 had loose bolts in its suspension, repaired by a precision robotic wrench.

- Car5 receives a solar-powered roof for eco-friendly VIP travel.

# Sample Output Screenshot

**Sample Output:**=== Assembly Order ===  
Assembling: Hood  
Assembling: Battery  
Assembling: Doors  
Assembling: Wheels  
Assembling: Chassis  
Assembling: Engine  
  
=== Garage Storage ===  
Car3 Car4 Car5 Car6 Car7 Car8 Car9 Car10   
  
=== Defective Tracker ===  
Defective Cars: Car3 -> Car6  
Repaired Car (Forward): Car3  
Repaired Car (Backward): Car3  
  
=== VIP Upgrade Checks ===  
Upgrade Check: Car1  
Upgrade Check: Car5  
Upgrade Check: Car1  
Upgrade Check: Car5