DSA Assignment Report

Assignment Title: Robotic Assembly Line Simulator using Data Structures

Name: BIBEK KUMAR MALIK

Roll Number: ME24B1044

**Problem Statement**

Design a simulator for a futuristic car manufacturing plant that uses robots to assemble car prototypes efficiently. The system should manage part deliveries, robot arm assembly tasks, storage of assembled prototypes, and tracking of defective and VIP vehicles using appropriate data structures.

# Key Objectives

• Simulate the arrival and processing of car parts.  
• Assemble car prototypes in a specific order.  
• Store prototypes in a garage with limited capacity.  
• Track defective and repaired vehicles.  
• Prioritize VIP cars for urgent upgrades.  
• Efficiently manage data using appropriate data structures.

# Design Explanation

## Why I Chose These Data Structures:

|  |  |  |
| --- | --- | --- |
| Data Structure | Subsystem | Reason |
| Queue | Part Delivery System | Follows First-In-First-Out (FIFO) to process parts as they arrive. |
| Stack | Robot Arm Task Manager | Follows Last-In-First-Out (LIFO) for assembly, ensuring finishing parts like the 'Hood' are added last. |
| Array | Assembly Storage Unit | Fixed-size storage for fast and simple management of completed prototypes. |
| Singly Linked List | Defective Prototype Tracker | Dynamic memory allocation for tracking defective cars without a fixed size. |
| Doubly Linked List | Repaired Car Tracker | Allows forward and backward traversal for detailed inspection after repair. |
| Circular Linked List | VIP Priority Upgrades | Enables continuous cycling through high-priority vehicles for regular upgrade checks. |

# Logic of the Code

1. Parts are enqueued into a queue.  
2. Robot arm dequeues parts and pushes them onto a stack (LIFO).  
3. Parts are popped from the stack to simulate assembly in reverse order.  
4. Assembled prototypes are stored in an array.  
 - If the garage is full, the oldest prototype is removed (shift operation) before adding a new one.  
5. Defective cars ('Car3' and 'Car6') are added to a singly linked list.  
 - 'Car3' is later removed and moved to a doubly linked list for detailed inspection.  
6. VIP cars ('Car1' and 'Car5') are added to a circular linked list.  
 - The list is traversed in cycles for continuous upgrade checks.

# Sample Output

=== Part Delivery System (Queue) ===  
Enqueued part: Engine  
Enqueued part: Chassis  
Enqueued part: Wheels  
Enqueued part: Doors  
Enqueued part: Battery  
Enqueued part: Hood  
  
=== Robot Arm Task Manager (Stack) ===  
Dequeued part: Engine → Pushed to stack  
Dequeued part: Chassis → Pushed to stack  
Dequeued part: Wheels → Pushed to stack  
Dequeued part: Doors → Pushed to stack  
Dequeued part: Battery → Pushed to stack  
Dequeued part: Hood → Pushed to stack  
  
Assembly Order (from stack — LIFO):  
Hood  
Battery  
Doors  
Wheels  
Chassis  
Engine  
  
=== Assembly Storage Unit (Garage) ===  
Added: Car1  
Added: Car2  
Added: Car3  
Added: Car4  
Added: Car5  
Added: Car6  
Added: Car7  
Added: Car8  
Garage full! Removing: Car1  
Added: Car9  
Garage full! Removing: Car2  
Added: Car10  
  
Garage content:  
Car3, Car4, Car5, Car6, Car7, Car8, Car9, Car10  
  
=== Defective Prototype Tracker ===  
Defective: Car3  
Defective: Car6  
  
After Repair:  
Car3 moved to repaired list.  
  
Repaired List (Forward): Car3  
Repaired List (Backward): Car3  
  
=== VIP Priority Upgrades ===  
VIP Added: Car1  
VIP Added: Car5  
  
Two rounds of VIP upgrade check:  
Upgrade check on: Car1  
Upgrade check on: Car5  
Upgrade check on: Car1  
Upgrade check on: Car5

**C CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 10 //CONSTANT

#define GARAGE\_CAPACITY 8

typedef struct { //QUEUE FOR PART DELIVERY SYSTEM

    char parts[SIZE][20];

  int front, rear;

} Queue;

void initQueue(Queue \*q) {

    q->front = q->rear = -1;

}

int isQueueFull(Queue \*q) {

    return q->rear == SIZE - 1;

}

int isQueueEmpty(Queue \*q) {

    return q->front == -1;

}

void enqueue(Queue \*q, char part[]) {

    if (isQueueFull(q)) {

        printf("Queue is full!\n");

        return;

    }

    if (isQueueEmpty(q)) q->front = 0;

    strcpy(q->parts[++q->rear], part);

}

void dequeue(Queue \*q, char part[]) {

    if (isQueueEmpty(q)) {

        printf("Queue is empty!\n");

        return;

    }

  strcpy(part, q->parts[q->front++]);

    if (q->front > q->rear) initQueue(q);

}

typedef struct { //STACK FOR MANAGING THE ROBOT ARM TAK

    char items[SIZE][20];

    int top;

} Stack;

void initStack(Stack \*s) {

    s->top = -1;

}

int isStackFull(Stack \*s) {

    return s->top == SIZE - 1;

}

int isStackEmpty(Stack \*s) {

    return s->top == -1;

}

void push(Stack \*s, char item[]) {

    if (isStackFull(s)) {

        printf("Stack is full!\n");

        return;

    }

    strcpy(s->items[++s->top], item);

}

void pop(Stack \*s, char item[]) {

    if (isStackEmpty(s)) {

        printf("Stack is empty!\n");

        return;

    }

    strcpy(item, s->items[s->top--]);

}

char garage[GARAGE\_CAPACITY][20]; //ARRAY IS THE ASSEMBLY STORAGE UNIT

int garageCount = 0;

void addPrototype(char prototype[]) {

    if (garageCount < GARAGE\_CAPACITY) {

        strcpy(garage[garageCount++], prototype);

    } else {

        // Shift left to remove oldest

        for (int i = 0; i < GARAGE\_CAPACITY - 1; i++)

            strcpy(garage[i], garage[i + 1]);

        strcpy(garage[GARAGE\_CAPACITY - 1], prototype);

    }

}

void displayGarage() {

    printf("Garage:\n");

    for (int i = 0; i < garageCount; i++)

        printf("%s\n", garage[i]);

}

typedef struct Node { //USING LINKED LIST TO DETECT DEFECTIVE PART

    char car[20];

    struct Node \*next;

} Node;

Node \*defectiveHead = NULL;

void addDefective(char car[]) {

    Node \*temp = (Node\*)malloc(sizeof(Node));

    strcpy(temp->car, car);

    temp->next = defectiveHead;

    defectiveHead = temp;

}

void removeDefective(char car[]) {

    Node \*temp = defectiveHead, \*prev = NULL;

    while (temp) {

        if (strcmp(temp->car, car) == 0) {

            if (prev) prev->next = temp->next;

            else defectiveHead = temp->next;

            free(temp);

            return;

        }

        prev = temp;

        temp = temp->next;

    }

}

typedef struct DNode { // Doubly Linked List forRepaired Tracker

    char car[20];

    struct DNode \*next, \*prev;

} DNode;

DNode \*repairedHead = NULL, \*repairedTail = NULL;

void addRepaired(char car[]) {

    DNode \*temp = (DNode\*)malloc(sizeof(DNode));

    strcpy(temp->car, car);

    temp->next = NULL;

    temp->prev = repairedTail;

    if (repairedTail) repairedTail->next = temp;

    else repairedHead = temp;

  repairedTail = temp;

}

void traverseRepairedForward() {

    DNode \*temp = repairedHead;

    printf("Repaired (forward):\n");

    while (temp) {

        printf("%s\n", temp->car);

        temp = temp->next;

    }

}

void traverseRepairedBackward() {

    DNode \*temp = repairedTail;

    printf("Repaired (backward):\n");

    while (temp) {

        printf("%s\n", temp->car);

        temp = temp->prev;

    }

}

typedef struct CNode { // Circular Linked List for VIP Upgrades.

char car[20];

    struct CNode \*next;

} CNode;

CNode \*vipHead = NULL;

void addVIP(char car[]) {

    CNode \*temp = (CNode\*)malloc(sizeof(CNode));

    strcpy(temp->car, car);

    if (!vipHead) {

        vipHead = temp;

        temp->next = vipHead;

    } else {

      CNode \*p = vipHead;

        while (p->next != vipHead) p = p->next;

        p->next = temp;

        temp->next = vipHead;

    }

}

void traverseVIP(int rounds) {

    if (!vipHead) return;

    CNode \*temp = vipHead;

    printf("VIP Upgrade Cycles:\n");

    for (int i = 0; i < rounds; i++) {

        printf("%s\n", temp->car);

        temp = temp->next;

    }

}

int main() {

    Queue q;

    Stack s;

    char part[20];

    char \*parts[] = {"Engine", "Chassis", "Wheels", "Doors", "Battery", "Hood"};

    char prototype[20];

  initQueue(&q);

    initStack(&s);

// Part Delivery & Robot Arm

    printf("Enqueueing parts:\n");

    for (int i = 0; i < 6; i++) {

        enqueue(&q, parts[i]);

        printf("%s added to queue\n", parts[i]);

    }

printf("\nTransferring parts to stack:\n");

    while (!isQueueEmpty(&q)) {

        dequeue(&q, part);

        push(&s, part);

        printf("%s moved to stack\n", part);

    }

printf("\nAssembly Order (LIFO):\n");

    while (!isStackEmpty(&s)) {

        pop(&s, part);

        printf("%s assembled\n", part);

    }

  // Assembly Storage Unit

    printf("\nAdding prototypes to garage:\n");

    for (int i = 1; i <= 10; i++) {

        sprintf(prototype, "Car%d", i);

        addPrototype(prototype);

    }

    displayGarage();

// Defective Prototype Tracker

    printf("\nTracking defective cars:\n");

    addDefective("Car3");

    addDefective("Car6");

removeDefective("Car3");

    addRepaired("Car3");

    traverseRepairedForward();

    traverseRepairedBackward();

// VIP Priority Upgrades

    printf("\nVIP Upgrade List:\n");

    addVIP("Car1");

    addVIP("Car5");

    traverseVIP(4);

return 0;}