**Robotic Assembly Line Simulator**

**NAME: N. Amulya**

**ROLL NO.: ME24I1025**

The key objective of this assignment is to **design and simulate a Robotic Assembly Line** using fundamental **data structures**

The **Robotic Assembly Line** aims to simulate **automated car manufacturing**, where robots handle part delivery, assembly, defect tracking, and upgrades efficiently using **structured data management**.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 6 // Maximum number of car parts in the queue

#define GARAGEC 8

// Structure for singly linked list node

struct Node {

char \*data;

struct Node \*next;

};

// Structure for doubly linked list node

struct DNode {

char \*data;

struct DNode \*prev;

struct DNode \*next;

};

// Function to allocate memory

void \*safe\_malloc(size\_t size) {

void \*ptr = malloc(size);

if (!ptr) {

printf("Memory allocation failed!\n");

exit(1); // Exit if memory allocation fails

}

return ptr;

}

//main function

int main() {

// Queue representing car parts

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

char \*stack[MAX]; //Using Stack to hold parts

int front = 0, rear = MAX, top = -1;

// Garage storing completed prototypes

char \*garage[GARAGEC] = {NULL};

char \*prototypes[10] = {"Car1", "Car2", "Car3", "Car4", "Car5", "Car6", "Car7", "Car8", "Car9", "Car10"};

int count = 0;

// Transfering queue contents to stack

while (front < rear) {

stack[++top] = queue[front++];

}

// Displaying assembly order

printf("Assembly Order:\n");

while (top >= 0) {

printf("%s\n", stack[top--]);

}

// Simulate adding prototypes to the garage

for (int i = 0; i < 10; i++)

{

if (count < GARAGEC)

{

garage[count++] = prototypes[i];

} else {

for (int j = 1; j < GARAGEC; j++)

{

garage[j - 1] = garage[j];

}

garage[GARAGEC - 1] = prototypes[i];

}

}

// Displaying updated contents

printf("Garage Contents:\n");

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

printf("%s\n", garage[i]);

}

// Singly linked list for defective prototypes

struct Node \*head = safe\_malloc(sizeof(struct Node));

head->data = "Car3";

head->next = safe\_malloc(sizeof(struct Node));

head->next->data = "Car6";

head->next->next = NULL;

// Moving "Car3" to a doubly linked list

struct DNode \*d\_head = safe\_malloc(sizeof(struct DNode));

d\_head->data = "Car3";

d\_head->prev = NULL;

d\_head->next = NULL;

// Forward traversal of doubly linked list

struct DNode \*current = d\_head;

printf("Doubly Linked List Forward:\n");

while (current) {

printf("%s\n", current->data);

current = current->next;

}

// Backward traversal of doubly linked list

current = d\_head;

printf("Doubly Linked List Backward:\n");

while (current) {

printf("%s\n", current->data);

current = current->prev;

}

//Using Circular linked list for VIP upgrade prototypes

struct Node \*car1 = safe\_malloc(sizeof(struct Node));

struct Node \*car5 = safe\_malloc(sizeof(struct Node));

car1->data = "Car1";

car5->data = "Car5";

car1->next = car5;

car5->next = car1;

// Traversing circular linked list twice

struct Node \*circular\_current = car1;

printf("Circular Linked List Traversal:\n");

for (int i = 0; i < 4; i++) {

printf("%s\n", circular\_current->data);

circular\_current = circular\_current->next;

}

// using free allocated memory to prevent memory leaks

free(head->next);

free(head);

free(d\_head);

free(car1);

free(car5);

return 0;

}

Output:

Assembly Order:

Hood

Battery

Doors

Wheels

Chassis

Engine

Garage Contents:

Car3

Car4

Car5

Car6

Car7

Car8

Car9

Car10

Doubly Linked List Forward:

Car3

Doubly Linked List Backward:

Car3

Circular Linked List Traversal:

Car1

Car5

Car1

Car5

=== Code Execution Successful ===