//Code , RISHAB R BABU , ME24I1029

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Queue Implementation
struct QueueNode {
    char item[20];
    struct QueueNode* next;
};

struct Queue {
    struct QueueNode* front;
    struct QueueNode* rear;
};

void enqueue(struct Queue* q, const char* item) {
    struct QueueNode* newNode = (struct QueueNode*)malloc(sizeof(struct
QueueNode));
    strcpy(newNode->item, item);
    newNode->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}
```

```
// Stack Implementation
struct StackNode {
    char item[20];
    struct StackNode* next;
};

void push(struct StackNode** stack, const char* item) {
    struct StackNode* newNode = (struct StackNode*)malloc(sizeof(struct
StackNode));
    strcpy(newNode->item, item);
    newNode->next = *stack;
    *stack = newNode;
}

void dequeueToStack(struct Queue* q, struct StackNode** stack) {
    while (q->front != NULL) {
        struct QueueNode* temp = q->front;
        push(stack, temp->item);
        q->front = q->front->next;
        free(temp);
    }
    q->rear = NULL;
}

void displayDispatch(struct StackNode** stack) {
    while (*stack) {
        printf("Dispatched: %s\n", (*stack)->item);
        struct StackNode* temp = *stack;
```

```
        *stack = (*stack)->next;
        free(temp);
    }
}

// Flight Log
struct FlightLog {
    char** entries;
    int capacity;
    int count;
};

void initFlightLog(struct FlightLog* log, int capacity) {
    log->entries = malloc(sizeof(char*) * capacity);
    for (int i = 0; i < capacity; i++)
        log->entries[i] = malloc(20);
    log->capacity = capacity;
    log->count = 0;
}

void logDelivery(struct FlightLog* log, const char* delivery) {
    if (log->count < log->capacity) {
        strcpy(log->entries[log->count++], delivery);
    } else {
        for (int i = 1; i < log->capacity; i++)
            strcpy(log->entries[i - 1], log->entries[i]);
        strcpy(log->entries[log->capacity - 1], delivery);
    }
}
```

```
void displayFlightLog(struct FlightLog* log) {
    printf("\nFlight Log:\n");
    for (int i = 0; i < log->count; i++)
        printf("%s\n", log->entries[i]);
}

void freeFlightLog(struct FlightLog* log) {
    for (int i = 0; i < log->capacity; i++)
        free(log->entries[i]);
    free(log->entries);
}

// Overloaded Drones (Singly Linked List)
struct SinglyNode {
    char drone_id[20];
    struct SinglyNode* next;
};

void addSinglyNode(struct SinglyNode** head, const char* drone_id) {
    struct SinglyNode* newNode = (struct SinglyNode*)malloc(sizeof(struct
SinglyNode));
    strcpy(newNode->drone_id, drone_id);
    newNode->next = *head;
    *head = newNode;
}

// Serviced Drones (Doubly Linked List)
struct DoublyNode {
```

```
    char drone_id[20];
    struct DoublyNode* next;
    struct DoublyNode* prev;
};

void moveToDoubly(struct SinglyNode** singlyHead, struct DoublyNode**
doublyHead) {
    if (*singlyHead == NULL) return;
    struct SinglyNode* temp = *singlyHead;
    *singlyHead = temp->next;

    struct DoublyNode* newNode = (struct DoublyNode*)malloc(sizeof(struct
DoublyNode));
    strcpy(newNode->drone_id, temp->drone_id);
    newNode->next = *doublyHead;
    newNode->prev = NULL;
    if (*doublyHead != NULL)
        (*doublyHead)->prev = newNode;
    *doublyHead = newNode;

    free(temp);
}

void traverseDoubly(struct DoublyNode* head) {
    struct DoublyNode* last = NULL;
    printf("\nForward: ");
    while (head) {
        printf("%s ", head->drone_id);
        last = head;
        head = head->next;
```

```
    }
    printf("\nBackward: ");
    while (last) {
        printf("%s ", last->drone_id);
        last = last->prev;
    }
    printf("\n");
}

// Emergency Reroute (Circular Linked List)
struct CircularNode {
    char drone_id[20];
    struct CircularNode* next;
};

void addCircularNode(struct CircularNode** head, const char* drone_id) {
    struct CircularNode* newNode = (struct CircularNode*)malloc(sizeof(struct
CircularNode));
    strcpy(newNode->drone_id, drone_id);
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
    } else {
        struct CircularNode* temp = *head;
        while (temp->next != *head)
            temp = temp->next;
        temp->next = newNode;
        newNode->next = *head;
    }
```

```
}

void traverseCircular(struct CircularNode* head) {
    if (head == NULL) return;
    struct CircularNode* temp = head;
    int count = 0;
    printf("\nEmergency Reroute: ");
    do {
        printf("%s ", temp->drone_id);
        temp = temp->next;
        count++;
    } while (temp != head && count < 4);
    printf("\n");
}

// Main Function
int main() {
    struct Queue deliveryQueue = { NULL, NULL };
    struct StackNode* dispatchStack = NULL;
    struct FlightLog flightLog;
    struct SinglyNode* overloaded = NULL;
    struct DoublyNode* serviced = NULL;
    struct CircularNode* reroute = NULL;

    initFlightLog(&flightLog, 6);

    int choice;
    char drone_id[20];
    while (1) {
```

```
        printf("\n1. Queue & Dispatch Deliveries\n2. Log Deliveries\n3. Add Overloaded Drone\n4. Service Drone\n5. View Maintenance Log\n6. Add Emergency Reroute\n7. View Emergency Route\n0. Exit\nChoice: ");
        scanf("%d", &choice);


        switch (choice) {
            case 1:
                enqueue(&deliveryQueue, "Food");
                enqueue(&deliveryQueue, "Medicine");
                enqueue(&deliveryQueue, "Fuel");
                enqueue(&deliveryQueue, "Spare Parts");
                enqueue(&deliveryQueue, "Tools");
                enqueue(&deliveryQueue, "Water");
                dequeueToStack(&deliveryQueue, &dispatchStack);
                displayDispatch(&dispatchStack);
                break;
            case 2:
                for (int i = 1; i <= 8; i++) {
                    sprintf(drone_id, "Del%d", i);
                    logDelivery(&flightLog, drone_id);
                }
                displayFlightLog(&flightLog);
                break;
            case 3:
                printf("Enter Drone ID: ");
                scanf("%s", drone_id);
                addSinglyNode(&overloaded, drone_id);
                break;
            case 4:
                moveToDoubly(&overloaded, &serviced);
```

```
            break;
        case 5:
            traverseDoubly(serviced);
            break;
        case 6:
            printf("Enter Drone ID for Emergency Reroute: ");
            scanf("%s", drone_id);
            addCircularNode(&reroute, drone_id);
            break;
        case 7:
            traverseCircular(reroute);
            break;
        case 0:
            freeFlightLog(&flightLog);
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice.\n");
      }
   }
   return 0;
}
```

Name: Rishab R Babu Roll No: ME2451029

Explanation of Question

To implement Cargo drone traffic controller following steps are necessary

This program simulates a comprehensive drone delivery management system using various data structures to handle different scenarios. The core components include queue, stack, singly linked list, doubly linked list, circular linked list, and a fixed-size array for logging.

The queue is used to manage delivery requests in the order they are received (FIFO - First In, First Out). Whenever a delivery item is added, it is enqueued at the rear. To dispatch the delivery, the item at the front of the queue is dequeued and pushed onto a stack. This stack ensures that the last delivery request becomes the first to be dispatched, mimicking urgency or priority (LIFO - Last In, First Out). Once all items are in the stack, they are popped out and displayed as dispatched deliveries.

A fixed-size 2D array acts as a flight log that stores the last six delivery entries. Each time a delivery is dispatched, it is logged. If the log already contains six entries, the oldest entry is removed, and the new one is added at the end, effectively sliding the array to make room.

For drones that become overloaded or require repairs, a singly linked list is used to track them. New drones are added at the beginning of the list to keep recent entries on top. Once these drones are serviced, they are moved to a doubly linked list, which allows viewing the history in both forward and backward directions, providing flexibility in tracking service records.

Emergency drones that follow circular routes are managed using a circular linked list. Each time a new emergency drone is added, it is inserted after the head. During traversal, the program only displays four drones to simulate a portion of the infinite circular path without entering an endless loop.

The main function offers a user-interactive menu with options to request and dispatch deliveries, log activities, mark drones as overloaded, service drones, and display emergency routes. Each selection triggers the appropriate data structure operation, and memory is managed dynamically to ensure clean operation.

This program demonstrates how different data structures are suitable for specific real-world problems and how they can be integrated into a unified system for simulation and operational control.

Why Struct was used?

It allows us to aggregate data type and use pointers better which reduces data traverse time without using new variable every time , for 2D array is used in log part

Why Malloc?

Malloc allows dynamic memory allocation that saves space and add and remove new entries without overflow or underflow

struct Queue { here the basic structure of code is explained a queue

struct QueueNode* front; queue requires two pointers are used to make sure

struct QueueNode* rear; FIFO is obeyed as front pointer points to element

}; that is to be dequeued first both are necessary to

Ensure smooth traversing.

struct StackNode {

char item[20]; struct StackNode* next; 2D array to store data (item) and node

};

The struct SinglyNode is necessary to represent drones marked as overloaded or damaged in a singly linked list, where drones are added quickly to the front, enabling fast updates.

struct SinglyNode {

char drone_id[20]; struct SinglyNode* next;

};

Once these drones are repaired, they are moved to a more advanced structure: struct DoublyNode which allows both forward and backward tracking of service history for maintenance of logs

struct DoublyNode {

char drone_id[20]; struct DoublyNode* next; struct DoublyNode* prev;

};

CircularNode used to rotate emergency drones repeadly like it starts with one drone then to next and then continue

struct CircularNode {

char drone_id[20]; struct CircularNode* next;

};

Item part

These functions implement a queue-to-stack transfer system for drone dispatch. The `enqueue()` function adds delivery requests to a FIFO queue. `dequeueToStack()` simultaneously dequeues from the queue and pushes onto a LIFO stack (reversing

the order). Finally, `displayDispatch()` pops and displays items from the stack, showing the dispatch order where the last request ("Fuel") gets processed first. This LIFO approach prioritizes urgent deliveries that arrived last, like emergency fuel requests. Memory is properly managed with malloc/free throughout.

Log registry part

1) logDelivery()` manages a fixed-size log (6 entries) using a circular buffer approach. When full, it overwrites the oldest entry. `displayFlightLog()` shows all current entries with their position numbers. This implements a rolling log that automatically archives old deliveries.
2) `addSinglyNode()` prepends overloaded drones to a singly-linked list.
3) `moveToDoubly()` transfers a drone to a doubly-linked list after servicing, maintaining proper forward/backward links.
4) `traverseDoubly()` demonstrates bidirectional traversal capability of the serviced drones list.
5) `addCircularNode()` inserts drones into a circular linked list for continuous monitoring.
6) `traverseCircular()` shows two full cycles of the emergency routing path (4 nodes displayed for 2 drones), highlighting the endless loop nature of circular lists for real-time tracking.

Main part

The main function includes all the functions

Uses while loop and switch to give the user options and make it more interactive

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute/7. View Emergency Route/0. Exit

Choice: 1

Dispatched: Water

Dispatched: Tools

Dispatched: Spare Parts

Dispatched: Fuel

Dispatched: Medicine

Dispatched: Food

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute/7. View Emergency Route/0. Exit

Choice: 3

Enter Drone ID: 6

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute/7. View Emergency Route/0. Exit

Choice: 2

Flight Log:

Del3

Del4

Del5

Del6

Del7

Del8

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 3

Enter Drone ID: 14

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 4

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 5

Forward: 14

Backward: 14

1.Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 6

Enter Drone ID for Emergency Reroute: 6

1. Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 7

Emergency Reroute: 6

1.Queue & Dispatch Deliveries/2. Log Deliveries/3. Add Overloaded Drone

/4. Service Drone/5. View Maintenance Log/6. Add Emergency Reroute

/7. View Emergency Route/0. Exit

Choice: 0

Exiting...

=== Code Execution Successful ===