Elementary data Structures Assignment

S Balasubramanian

ME24B1018

Scenario: You're designing a Cargo Drone Traffic Controller to manage drone deliveries (e.g., "Food", "Medicine", "Tools", "Water", "Parts") in a busy airspace. The system uses data structures to optimize drone routes:

- **Delivery Request System (Queue):** Delivery requests queue up from ground stations.
- **Priority Dispatcher (Stack):** Urgent deliveries stack in LIFO order for immediate dispatch.
- **Flight Log Unit (Array):** Completed deliveries log into an array-based record (size: 6 slots). If full, the oldest log is archived.
- **Maintenance Tracker (Linked Lists):**
    - Overloaded drones go to a singly linked list.
    - Serviced drones move to a doubly linked list for dual-direction review.
    - Emergency drones cycle in a circular linked list for urgent rerouting.

Objective: Simulate drone traffic management, logging, and maintenance.

Tasks:

a) **Request and Dispatch**

- Simulate 6 delivery requests (e.g., "Food", "Medicine", "Tools", "Water", "Parts", "Fuel") arriving in a queue.
- Stack urgent deliveries in LIFO order. Write pseudocode or an algorithm to:
    - Enqueue all 6 requests.
    - Dequeue and push onto a stack.
    - Pop to show dispatch order.
- *Creativity Bonus:* Describe (in 2-3 sentences) why LIFO suits urgency (e.g., "Fuel" last to refuel drones mid-flight).

Request and Dispatch (Queue + Stack Simulation)

Explanation

- **Queue** (FIFO): Used for handling incoming delivery requests.
- **Stack** (LIFO): Urgent deliveries are dispatched in reverse order of request (last urgent in = first dispatched).

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 6                //Defining size as 6
```

```
#define MAX_LEN 20          //to describe the max length to be 20

                                                                        //  Queue (FIFO)

typedef struct

 {

char items[SIZE][MAX_LEN];

int front, rear;

} Queue;                                                    //defining Queue as structure

void initQueue(Queue *q)                     //initiate queue

{

q->front = 0;                           //queue front as 0

q->rear = -1;                           //queue rear as -1

}

int isQueueFull(Queue *q)         //For full condition

{

return q->rear == SIZE - 1;        //condition

}

int isQueueEmpty(Queue *q)     //for empty condition

{

return q->front > q->rear;

}

void enqueue(Queue *q, char *item) {

if (!isQueueFull(q)) {

q->rear++;

strcpy(q->items[q->rear], item);

}

}

char* dequeue(Queue *q)               //dequeue condition

 {

if (!isQueueEmpty(q))

 {

return q->items[q->front++];             //for empty check
}
return NULL;
}
                                                // Stack (LIFO)
typedef struct
 {
char items[SIZE][MAX_LEN];
int top;
} Stack;
void initStack(Stack *s)
{
s->top = -1;            //assign top as 1
}
int isStackFull(Stack *s)
 {
return s->top == SIZE - 1;        //for full
}
int isStackEmpty(Stack *s)
 {
return s->top == -1;          //for empty
}
void push(Stack *s, char *item)     //push
 {
if (!isStackFull(s))
{
strcpy(s->items[++(s->top)], item);   //copy to another
}
}
char* pop(Stack *s)
{
if (!isStackEmpty(s))
```

```
{

return s->items[(s->top)--];

}

return NULL;

}

                                                    // Main Simulation

void requestAndDispatch()

{

Queue q;

Stack s;

char *deliveries[SIZE] = {"Food", "Medicine", "Tools", "Water", "Parts", "Fuel"};

initQueue(&q);

initStack(&s);

// Enqueue all 6 requests

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

 {

enqueue(&q, deliveries[i]);

}

                          // Dequeue and push onto stack

while (!isQueueEmpty(&q))

{

char *item = dequeue(&q);

push(&s, item);

}

                                                // Pop from stack to show dispatch order

printf("Urgent Dispatch Order (LIFO):\n");

while (!isStackEmpty(&s))

 {

printf("%s\n", pop(&s));

}

}
```

b) **Flight Log Unit**

- Log deliveries in a 6-slot array.
- Simulate logging 8 deliveries (e.g., "Del1", "Del2", ..., "Del8"). If full, archive the oldest. Write pseudocode or an algorithm to:
    - Insert the first 6 deliveries.
    - Handle overflow for "Del7" and "Del8".
- *Creativity Bonus:* Suggest (in 2-3 sentences) a reason for archiving (e.g., airspace regulation compliance).

Flight Log Unit (Array with Overflow Handling)

```
#define LOG_SIZE 6

void flightLogSimulation()

 {

char *log[LOG_SIZE] = {NULL};  // Array of delivery logs

char *deliveries[8] = {"Del1", "Del2", "Del3", "Del4", "Del5", "Del6", "Del7", "Del8"};

int start = 0;                                   //                 Points to the oldest log (for overwriting)

for (int i = 0; i < 8; i++)

 {

int pos = (start + i) % LOG_SIZE;

log[pos] = deliveries[i];

                                                       // If more than 6, start archiving

if (i >= LOG_SIZE)

 {

start = (start + 1) % LOG_SIZE;

}

printf("Logged: %s\n", deliveries[i]);

}

printf("\nCurrent Log:\n");

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

 {

printf("%s\n", log[(start + i) % LOG_SIZE]);

}

}
```

c) **Overloaded Drone Tracker**

- "Drone3" and "Drone6" are overloaded. Add to a singly linked list.
- Move "Drone3" to a doubly linked list post-service. Write pseudocode or an algorithm to:

Overloaded Drone Tracker (Singly & Doubly Linked Lists)

```
                        //   Singly Linked List

typedef struct Node {

char name[10];

struct Node *next;

} Node;


                          // - Doubly Linked List

typedef struct DNode

 {

char name[10];

struct DNode *prev;

struct DNode *next;

} DNode;


Node *overloaded = NULL;

DNode *serviced = NULL;


void insertSingly(char *name)

 {

Node *newNode = (Node *)malloc(sizeof(Node));           //DMA

strcpy(newNode->name, name);               //for name

newNode->next = overloaded;

overloaded = newNode;

}
```

```
void moveToDoubly(char *name) {
Node *temp = overloaded, *prev = NULL;

        // Remove from singly list
while (temp != NULL && strcmp(temp->name, name) != 0) {
prev = temp;
temp = temp->next;
}

if (temp) {
if (prev) prev->next = temp->next;
else overloaded = temp->next;
// Insert into doubly list
DNode *dNode = (DNode *)malloc(sizeof(DNode));          //DMA
strcpy(dNode->name, name);
dNode->next = serviced;
dNode->prev = NULL;
if (serviced) serviced->prev = dNode;
serviced = dNode;
free(temp);
}
}
void traverseDoubly()
 {
DNode *curr = serviced;
printf("Forward:\n");
while (curr)
 {
printf("%s\n", curr->name);
if (!curr->next) break;
curr = curr->next;
}

printf("Backward:\n");

while (curr)

 {

printf("%s\n", curr->name);

curr = curr->prev;

}

}
```

d) **Emergency Rerouting**

- "Drone1" and "Drone4" need urgent rerouting (e.g., storm avoidance). Add to a circular linked list. Write pseudocode or an algorithm to:
    - Insert "Drone1" and "Drone4".
    - Traverse twice.
- *Creativity Bonus:* Invent (in 2-3 sentences) a rerouting tweak (e.g., "Drone4 gets a wind-resistant shell").

```
                    //Emergency Rerouting (Circular Linked List)

typedef struct CNode

{

char name[10];

struct CNode *next;

} CNode;

CNode *emergencyHead = NULL;

void insertCircular(char *name)

 {

CNode *newNode = (CNode *)malloc(sizeof(CNode));          //DMA

strcpy(newNode->name, name);

if (emergencyHead == NULL)

{

emergencyHead = newNode;

newNode->next = newNode;

}
else
 {
CNode *temp = emergencyHead;
while (temp->next != emergencyHead) {
temp = temp->next;
}
temp->next = newNode;
newNode->next = emergencyHead;
}
}
void traverseCircular(int times) {
if (!emergencyHead) return;
CNode *curr = emergencyHead;
int count = 0;
printf("Emergency Reroute Drones:\n");
while (count < times)       //condition
 {
printf("%s\n", curr->name);
curr = curr->next;
if (curr == emergencyHead) count++;
}
}
```

Putting It All Together

```
int main()
{
printf("=== A. Request and Dispatch ===\n");
requestAndDispatch();
printf("\n=== B. Flight Log ===\n");
flightLogSimulation();
printf("\n=== C. Overloaded Drones ===\n");
insertSingly("Drone3");
insertSingly("Drone6");

moveToDoubly("Drone3");

traverseDoubly();

printf("\n=== D. Emergency Rerouting ===\n");

insertCircular("Drone1");

insertCircular("Drone4");

traverseCircular(2);

return 0;

}
```

Introduction

The Cargo Drone Traffic Controller system is designed to efficiently manage drone deliveries such as "Food", "Medicine", "Tools", "Water", "Parts", and "Fuel" in a busy airspace.
It utilizes fundamental data structures — Queue, Stack, Array, and Linked Lists — to optimize the flow of delivery requests, dispatch urgent missions, maintain flight logs, and track drone maintenance and emergencies.

This simulation covers four main parts:

1. Request and Dispatch (Queue + Stack)
2. Flight Log Unit (Array)
3. Overloaded Drone Tracker (Singly and Doubly Linked Lists)
4. Emergency Rerouting (Circular Linked List)

Each part is coded carefully to reflect real-world airspace operations where drones need to be prioritized, logged, maintained, and rerouted based on conditions.

1. Request and Dispatch (Queue + Stack)

Concept:

- Queue (FIFO - First In, First Out) handles incoming delivery requests orderly.
- Stack (LIFO - Last In, First Out) manages urgent dispatches, ensuring the most recent critical delivery is prioritized.

Why LIFO for urgent dispatch?
Because emergencies often demand that the *last* reported urgent delivery (e.g., "Fuel") is dispatched *first*, ensuring faster resolution of the latest critical issue.

Code Explanation:

- A Queue is implemented with a structure containing an array and front and rear indices.
- Enqueue function adds new delivery requests.
- Dequeue function removes requests one by one.
- The dequeued items are pushed into a Stack.
- Pop operation from the Stack shows the dispatch order (latest urgent first).

Deliveries processed:
"Food" → "Medicine" → "Tools" → "Water" → "Parts" → "Fuel"
Resulting dispatch order: "Fuel", "Parts", "Water", "Tools", "Medicine", "Food".

2. Flight Log Unit (Array Simulation)

Concept:

- A fixed-size array (6 slots) is used to log completed deliveries.
- Overflow is handled by overwriting the oldest logs — similar to a circular buffer.
- This ensures the system maintains only the latest deliveries for regulatory compliance and efficient storage.

Why archive old logs?
Because airspace authorities require keeping only recent and relevant records, ensuring storage optimization without losing important operational data.

Code Explanation:

- An array of pointers is initialized to NULL.
- First 6 deliveries ("Del1" to "Del6") are inserted.
- When "Del7" and "Del8" arrive, they overwrite the oldest entries.
- Logging is cyclic, ensuring continuous updates without memory overflow.

3. Overloaded Drone Tracker (Linked Lists)

Concept:

- Overloaded drones are first added to a Singly Linked List.
- After maintenance/service, drones are moved into a Doubly Linked List for two-way traversal and easier monitoring.

Example:
"Drone3" and "Drone6" are initially overloaded.
After servicing "Drone3", it is transferred to the serviced list (doubly linked).

Code Explanation:

- A Singly Linked List (Node structure) manages drones awaiting service.
- The insertSingly() function inserts drones into the overloaded list.
- moveToDoubly() removes a node from the singly linked list and inserts it into a doubly linked list (DNode structure).

- traverseDoubly() prints the drones' names in both forward and reverse order, showcasing the two-way link.

4. Emergency Rerouting (Circular Linked List)

Concept:

- Emergency drones that need urgent rerouting (due to storms, technical failures, etc.) are managed using a Circular Linked List.
- This structure allows continuous monitoring and multiple traversals without ending — ideal for ongoing urgent situations.

Example:
"Drone1" and "Drone4" are inserted into the emergency rerouting list.

Code Explanation:

- Circular linked list (CNode structure) connects drones in a cycle.
- insertCircular() inserts drones into the list.
- traverseCircular(times) traverses the list a specific number of times (e.g., 2 rounds), allowing repeated monitoring without reinitialization.

Main Function (Bringing It All Together)

The main() function:

1. Simulates delivery request and dispatch.
2. Logs deliveries into an array with overflow handling.
3. Tracks overloaded drones and transitions them after service.
4. Handles emergency rerouting drones.

Each module is independent but integrates smoothly under the main controller.

Conclusion

This simulation project demonstrates:

- Practical application of fundamental data structures.

- Real-world adaptation for drone delivery airspace management.
- Handling normal operations, urgent situations, logging, and maintenance systematically.

It reflects good programming practice using Queue, Stack, Array, Singly Linked List, Doubly Linked List, and Circular Linked List for modular, efficient, and realistic control of autonomous delivery drones.