Implementation of Queue using Array and Linked List Implementation

Write a C program to implement a Queue using Array and linked List implementation and execute the following operation on stack.

(i) Enqueue

(ii) Dequeue

(iii) Display the elements in a Queue

PROGRAM:

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

#define MAX_SIZE 100

// Node structure for linked list implementation
typedef struct Node {
   int data;
   struct Node* next;
} Node;

// Structure for queue using linked list
typedef struct {
   Node *front, *rear;
} QueueLinkedList;

// Structure for queue using array
typedef struct {
   int items[MAX_SIZE];
   int front;
   int rear;
} QueueArray;
```

```
// Function prototypes for queue using array
void initQueueArray(QueueArray *q);
int isEmptyArray(QueueArray *q);
int isFullArray(QueueArray *q);
void enqueueArray(QueueArray *q, int value);
int dequeueArray(QueueArray *q);
void displayQueueArray(QueueArray *q);

// Function prototypes for queue using linked list
void initQueueLinkedList(QueueLinkedList *q);
int isEmptyLinkedList(QueueLinkedList *q);
void enqueueLinkedList(QueueLinkedList *q, int value);
int dequeueLinkedList(QueueLinkedList *q);
void displayQueueLinkedList(QueueLinkedList *q);

// Initialize queue using array
void initQueueArray(QueueArray *q) {
   q->front = -1;
   q->rear = -1;
}

// Check if queue using array is empty
int isEmptyArray(QueueArray *q) {
   return q->front == -1;
}

// Check if queue using array is full
int isFullArray(QueueArray *q) {
   return (q->rear + 1) % MAX_SIZE == q->front;
}

// Add element to queue using array (enqueue)
```

```
void enqueueArray(QueueArray *q, int value) {
    if (isFullArray(q)) {
        printf("Queue Overflow\n");
        return;
    }

    if (isEmptyArray(q)) {
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX_SIZE;
    q->items[q->rear] = value;
}

// Remove element from queue using array (dequeue)
int dequeueArray(QueueArray *q) {
    if (isEmptyArray(q)) {
        printf("Queue Underflow\n");
        return -1;
    }

    int removedValue = q->items[q->front];
    if (q->front == q->rear) {
        q->front = -1;
        q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX_SIZE;
    }
    return removedValue;
}

// Display elements of queue using array
void displayQueueArray(QueueArray *q) {
```

```
    if (isEmptyArray(q)) {
        printf("Queue is empty\n");
        return;
    }

    printf("Queue elements (Array): ");
    int i;
    for (i = q->front; i != q->rear; i = (i + 1) % MAX_SIZE) {
        printf("%d ", q->items[i]);
    }
    printf("%d\n", q->items[i]);
}

// Create a new node for linked list implementation
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Initialize queue using linked list
void initQueueLinkedList(QueueLinkedList *q) {
    q->front = q->rear = NULL;
}

// Check if queue using linked list is empty
int isEmptyLinkedList(QueueLinkedList *q) {
```

```
    return q->front == NULL;
}

// Add element to queue using linked list (enqueue)
void enqueueLinkedList(QueueLinkedList *q, int value) {
    Node* newNode = createNode(value);
    if (isEmptyLinkedList(q)) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

// Remove element from queue using linked list (dequeue)
int dequeueLinkedList(QueueLinkedList *q) {
    if (isEmptyLinkedList(q)) {
        printf("Queue Underflow\n");
        return -1;
    }

    Node* temp = q->front;
    int removedValue = temp->data;
    q->front = q->front->next;
    free(temp);

    if (q->front == NULL) {
        q->rear = NULL;
    }

    return removedValue;
}
```

```
// Display elements of queue using linked list
void displayQueueLinkedList(QueueLinkedList *q) {
    if (isEmptyLinkedList(q)) {
        printf("Queue is empty\n");
        return;
    }

    printf("Queue elements (Linked List): ");
    Node* temp = q->front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    QueueArray qArray;
    QueueLinkedList qLinkedList;

    // Initialize queues
    initQueueArray(&qArray);
    initQueueLinkedList(&qLinkedList);

    // Enqueue operations
    enqueueArray(&qArray, 10);
    enqueueArray(&qArray, 20);
    enqueueArray(&qArray, 30);

    enqueueLinkedList(&qLinkedList, 40);
    enqueueLinkedList(&qLinkedList, 50);
```

```
    enqueueLinkedList(&qLinkedList, 60);

    // Display elements
    displayQueueArray(&qArray);
    displayQueueLinkedList(&qLinkedList);

    // Dequeue operations
    printf("Dequeued element (Array): %d\n", dequeueArray(&qArray));
    printf("Dequeued element (Linked List): %d\n", dequeueLinkedList(&qLinkedList));

    // Display elements after dequeue
    displayQueueArray(&qArray);
    displayQueueLinkedList(&qLinkedList);

    return 0;
}
```

OUTPUT:

```
Queue elements (Array): 10 20 30
Queue elements (Linked List): 40 50 60
Dequeued element (Array): 10
Dequeued element (Linked List): 40
Queue elements (Array): 20 30
Queue elements (Linked List): 50 60
```