Experiment-5

Aim: Write a program to display, insert and delete element and remove duplicates to a circular queue using menu driven program. Also check for overflow and underflow condition.

Code:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX 5  // Define maximum size of the circular queue

// Circular Queue Structure
struct CircularQueue {
    int arr[MAX];
    int front;
    int rear;
};

// Function to initialize the queue
void initQueue(struct CircularQueue* queue) {
    queue->front = -1;
    queue->rear = -1;
}

// Function to check if the queue is full
bool isFull(struct CircularQueue* queue) {
    return (queue->front == (queue->rear + 1) % MAX);
}

// Function to check if the queue is empty
bool isEmpty(struct CircularQueue* queue) {
    return (queue->front == -1);
}

// Function to insert an element into the circular queue
void insert(struct CircularQueue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue overflow! Cannot insert %d\n", value);
```

```
        return;
    }

    if (isEmpty(queue)) {
        queue->front = 0;
    }

    queue->rear = (queue->rear + 1) % MAX;
    queue->arr[queue->rear] = value;
    printf("%d inserted into the queue.\n", value);
}

// Function to delete an element from the circular queue
int delete(struct CircularQueue* queue) {
    if (isEmpty(queue)) {
        printf("Queue underflow! Cannot delete\n");
        return -1;
    }

    int value = queue->arr[queue->front];

    if (queue->front == queue->rear) {
        // Queue has only one element
        queue->front = queue->rear = -1;
    } else {
        queue->front = (queue->front + 1) % MAX;
    }

    printf("%d deleted from the queue.\n", value);
    return value;
}

// Function to display the queue elements
void display(struct CircularQueue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements: ");
```

```
    int i = queue->front;
    while (i != queue->rear) {
        printf("%d ", queue->arr[i]);
        i = (i + 1) % MAX;
    }
    printf("%d\n", queue->arr[i]);  // Display the rear element
}

// Function to remove duplicates from the queue
void removeDuplicates(struct CircularQueue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty. No duplicates to remove.\n");
        return;
    }

    int i = queue->front;
    while (i != queue->rear) {
        int j = (i + 1) % MAX;
        while (j != queue->rear + 1) {
            if (queue->arr[i] == queue->arr[j]) {
                printf("Removing duplicate element %d\n", queue->arr[j]);
                // Shift elements to the left to remove duplicate
                int k = j;
                while (k != queue->rear) {
                    queue->arr[k] = queue->arr[(k + 1) % MAX];
                    k = (k + 1) % MAX;
                }
                queue->rear = (queue->rear - 1 + MAX) % MAX;
            } else {
                j = (j + 1) % MAX;
            }
        }
        i = (i + 1) % MAX;
    }

    printf("Duplicates removed.\n");
}

// Menu-driven program
int main() {
```

```
struct CircularQueue queue;
initQueue(&queue);

int choice, value;

while (1) {
    printf("\n*** Circular Queue Menu ***\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Display\n");
    printf("4. Remove Duplicates\n");
    printf("5. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to insert: ");
            scanf("%d", &value);
            insert(&queue, value);
            break;

        case 2:
            delete(&queue);
            break;

        case 3:
            display(&queue);
            break;

        case 4:
            removeDuplicates(&queue);
            break;

        case 5:
            printf("Exiting program.\n");
            return 0;

        default:
            printf("Invalid choice! Please try again.\n");
```

```
            }
        }

        return 0;
    }
```

Output: