Circular Queue operations using an array

Code:

```
 #include <stdio.h>
#include <stdlib.h>
#define MAX 5 // Maximum size of the circular queue

typedef struct {
   int data[MAX];
   int front;
   int rear;
} CircularQueue;

// Function prototypes
void initializeQueue(CircularQueue *q);
int isFull(CircularQueue *q);
int isEmpty(CircularQueue *q);
void insert(CircularQueue *q, int value);
int delete(CircularQueue *q);
void displayQueue(CircularQueue *q);
int getFront(CircularQueue *q);
int getRear(CircularQueue *q);

// Main function to test the circular queue
int main() {
   CircularQueue q;
   initializeQueue(&q);

   int choice, value;

   while (1) {
      printf("\nCircular Queue Operations:\n");
      printf("1. Insert\n");
      printf("2. Delete\n");
      printf("3. Display Queue\n");
      printf("4. Display Front Element\n");
      printf("5. Display Rear Element\n");
      printf("6. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            insert(&q, value);
            break;
         case 2:
            value = delete(&q);
            if (value != -1) {
               printf("Deleted value: %d\n", value);
            }
            break;
         case 3:
            displayQueue(&q);
            break;
         case 4:
            value = getFront(&q);
            if (value != -1) {
               printf("Front element: %d\n", value);
            }
            break;
         case 5:
            value = getRear(&q);
            if (value != -1) {
               printf("Rear element: %d\n", value);
            }
            break;
         case 6:
            exit(0);
```

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

    return 0;
}

// Initialize the circular queue
void initializeQueue(CircularQueue *q) {
    q->front = -1;
    q->rear = -1;
}

// Check if the circular queue is full
int isFull(CircularQueue *q) {
    return (q->rear + 1) % MAX == q->front;
}

// Check if the circular queue is empty
int isEmpty(CircularQueue *q) {
    return q->front == -1;
}

// Insert an element into the circular queue
void insert(CircularQueue *q, int value) {
    if (isFull(q)) {
        printf("Queue is full! Cannot insert.\n");
        return;
    }
    if (isEmpty(q)) { // First element being inserted
        q->front = 0;
    }
    q->rear = (q->rear + 1) % MAX;
    q->data[q->rear] = value;
    printf("Inserted %d into the queue.\n", value);
}

// Delete an element from the circular queue
int delete(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Cannot delete.\n");
        return -1;
    }
    int value = q->data[q->front];
    if (q->front == q->rear) { // Queue becomes empty
        q->front = q->rear = -1;
    } else {
        q->front = (q->front + 1) % MAX;
    }
    return value;
}

// Display all elements in the circular queue
void displayQueue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    int i = q->front;
    do {
        printf("%d ", q->data[i]);
        i = (i + 1) % MAX;
    } while (i != (q->rear + 1) % MAX);
    printf("\n");
}

// Get the front element of the circular queue
int getFront(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! No front element.\n");
        return -1;
```

```
    }
    return q->data[q->front];
}

// Get the rear element of the circular queue
int getRear(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! No rear element.\n");
        return -1;
    }
    return q->data[q->rear];
}
```

OUTPUT:

```
Circular Queue Operations:
1. Insert
2. Delete
3. Display Queue
4. Display Front Element
5. Display Rear Element
6. Exit
Enter your choice: 1
Enter the value to insert: 1
Inserted 1 into the queue.

Circular Queue Operations:
1. Insert
2. Delete
3. Display Queue
4. Display Front Element
5. Display Rear Element
6. Exit
Enter your choice: 1
Enter the value to insert: 2
Inserted 2 into the queue.

Circular Queue Operations:
1. Insert
2. Delete
3. Display Queue
4. Display Front Element
5. Display Rear Element
6. Exit
Enter your choice: 3
Queue elements: 1 2

Circular Queue Operations:
1. Insert
2. Delete
3. Display Queue
4. Display Front Element
5. Display Rear Element
6. Exit
Enter your choice: 1
Enter the value to insert: 3
Inserted 3 into the queue.
```