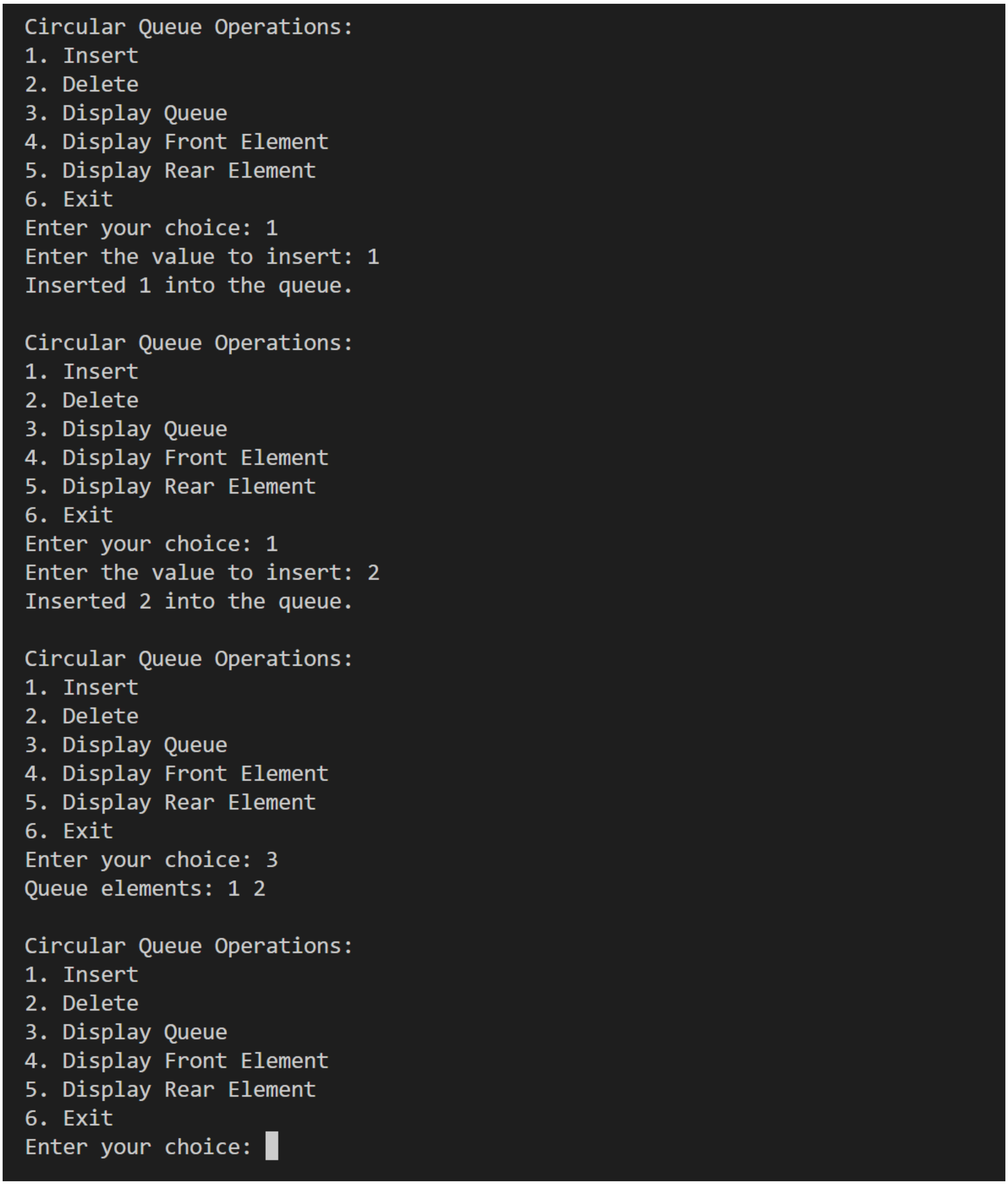
Circular Queue operations using a Linked List

<u>Code:</u>

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

typedef struct Node {
   int data;
   struct Node *next;
} Node;

typedef struct {
   Node *rear;
} CircularQueue;

void initializeQueue(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);

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;
}

void initializeQueue(CircularQueue *q) {
    q->rear = NULL;
}

int isEmpty(CircularQueue *q) {
    return q->rear == NULL;
}

void insert(CircularQueue *q, int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed! Cannot insert.\n");
        return;
    }
    newNode->data = value;
    if (isEmpty(q)) {
        newNode->next = newNode; // Points to itself in a single-node circular queue
        q->rear = newNode;
    } else {
        newNode->next = q->rear->next; // Link new node to the front
        q->rear->next = newNode;      // Rear points to the new node
        q->rear = newNode;            // Update rear to the new node
    }
    printf("Inserted %d into the queue.\n", value);
}

int delete(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! Cannot delete.\n");
        return -1;
    }
    Node *temp = q->rear->next; // The front node
    int value = temp->data;

    if (q->rear == temp) { // Single element in the queue
        q->rear = NULL;
    } else {
        q->rear->next = temp->next; // Update rear's next to skip the front node
    }

    free(temp);
    return value;
}

void displayQueue(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    Node *current = q->rear->next; // Start from the front node
    do {
        printf("%d ", current->data);
        current = current->next;
    } while (current != q->rear->next); // Traverse until we circle back to the front
    printf("\n");
}

int getFront(CircularQueue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty! No front element.\n");
        return -1;
    }
    return q->rear->next->data; // Front is the node pointed to by rear->next
}
```

```
int getRear(CircularQueue *q) {
   if (isEmpty(q)) {
      printf("Queue is empty! No rear element.\n");
      return -1;
   }
   return q->rear->data;
}
```

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: 
```