Batch: B1 **Roll No.: 1711072**

Experiment No. 4

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Deque using Circular Arrays

Objective: Implementation of double ended queue (deque) menu driven program.

Expected Outcome of Experiment:

CO	Outcome
CO1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein; (CLRS) "Introduction to Algorithms", Third Edition, The MIT Press.

Abstract:-

DATA STRUCTURE:

1. int queue[5] //array to store and manipulate queue of size 5 at max.
2. front = -1 //to keep track of front of the queue.
3. rear = -1 //to keep track of end of the queue.

OPERATIONS:

1. **Checking for empty queue**

   This function is used to check if the queue is empty or not.

2. **Checking for full queue**

   This function is used to check if the queue is empty or not.

3. **Display the queue elements**

   This function is used to display all the elements present in the queue from front to rear.

4. **Insert element from left (front)**

   This function is used to enter an element at the beginning of the queue.

5. **Insert element from right (rear)**

   This function is used to enter an element from the rear end of the queue.

6. **Delete element from left (front)**

   This function is used to delete an element from the front of the queue.

7. **Delete element from right (rear)**

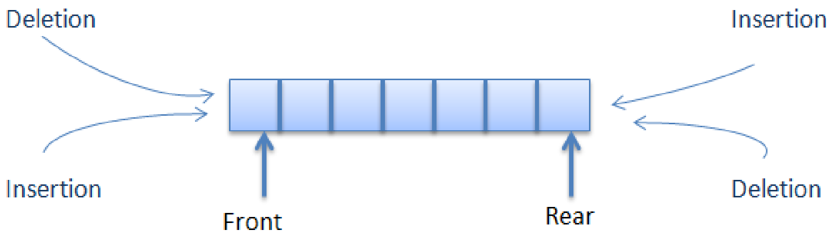
   This function is used to delete an element from the rear end of the queue.

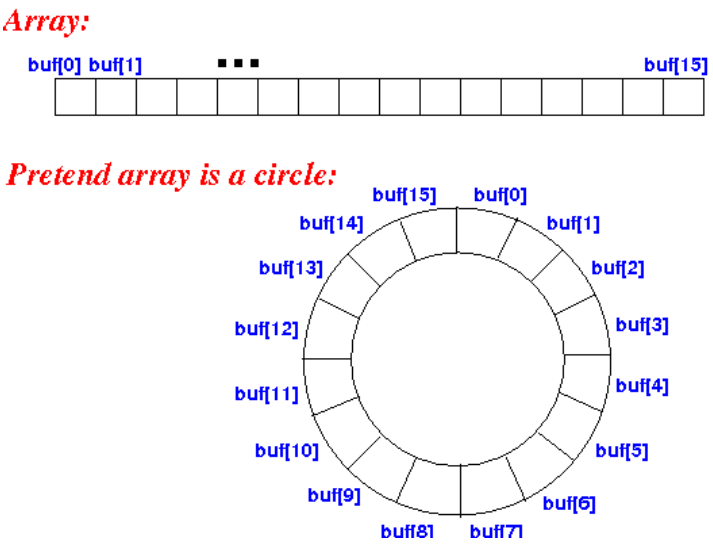
Related Theory: -

- A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO).
- There are many real life examples of queue:

  A good example of a queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- However, we are going to implement double ended queue in this experiment using circular array.
- Mainly the following three basic operations are performed in the deque:
  1. **insertFront()**: Adds an item at the front of Deque.
  2. **insertLast()**: Adds an item at the rear of Deque.
  3. **deleteFront()**: Deletes an item from front of Deque.
  4. **deleteLast()**: Deletes an item from rear of Deque.
  5. **getFront()**: Gets the front item from deque.
  6. **getRear()**: Gets the last item from deque.
  7. **isEmpty()**: Checks whether Deque is empty or not.
  8. **isFull()**: Checks whether Deque is full or not.
- Time Complexities of operations on deque:
  All operations take O (1) time. We do not run any loop in any of these operations.
- **Applications of Deque:**

  Since Deque supports both stack and queue operations, it can be used as both. The Deque data structure supports clockwise and anticlockwise rotations in O(1) time which can be useful in certain applications. Also, the problems where elements need to be removed and or added both ends can be efficiently solved using Deque. For example see Maximum of all subarrays of size k problem, 0-1 BFS and Find the first circular tour that visits all petrol pumps.
- **Implementation:**
  There are three ways to implement a deque:
  1. Using array
  2. **Using circular array**
  3. Using linked list





<u>Implementation Details</u>:

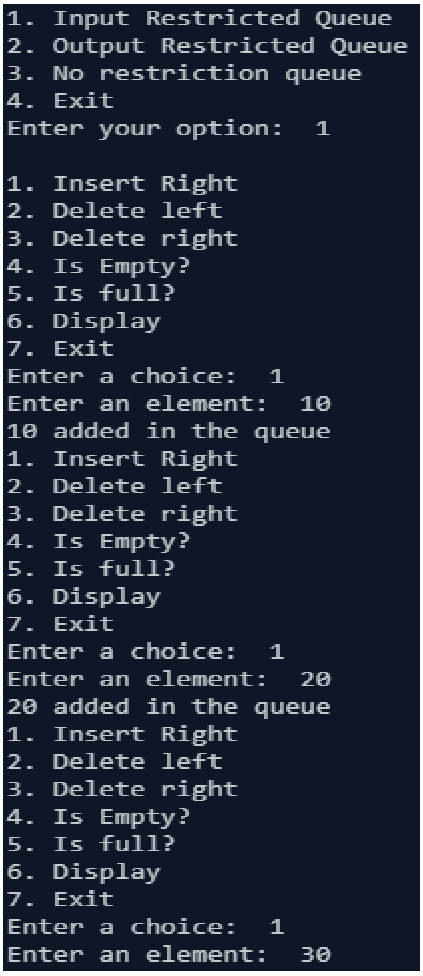
```
#include <stdio.h>
#include <stdlib.h>
int queue[5];
int front=-1, rear=-1, size=sizeof(queue)/sizeof(int);

int isEmpty(){
  if(front==-1 && rear==-1){
    //printf("Queue is empty.");
    return 1;
  }
  else{
    //printf("Queue isn't empty.");
    return 0;
  }
}
int isFull(){
  if(rear==(rear+1)%size){
    //printf("Queue is full!");
    return 1;
  }
  else{
    //printf("Queue isn't full");
    return 0;
  }
}
void display(){
  int i;
  if(isEmpty()){
    printf("Queue is empty.");
  }
  else{
    printf("The  element in queue is: ");
    for(i=front;i!=rear;i++)
    printf("%d ", queue[i]);
  }
  printf("%d", queue[i]);
}
void insert_right(int element){
  if(isEmpty()){
    front++;
    rear++;
    queue[rear]=element;
    printf("%d added in the queue", element);
  }
  else if((rear+1)%size==front){
    printf("Queue is full. Cannot add more");
    return;
  }
  else{
    rear=(rear+1)%size;
    queue[rear]=element;
    printf("%d added in the queue", element);
  }
}

void insert_left(int element){
  if(isEmpty()){
    front++;
    rear++;
    queue[front]=element;
    printf("%d added in the queue", element);
  }
  else if(front==(rear+1)%size){
    printf("Queue is full. Cannot add more");
    return;
  }
  else{
    front--;
    queue[front]=element;
    printf("%d added in the queue", element);
  }
}

void delete_right(){
  if(isEmpty()){
    printf("Cannot remove element as queue is already empty.");
    return;
  }
  else if(front==rear){
    front=-1;
    rear=-1;
  }
  else{
    rear=(rear-1)%size;
  }
}

void delete_left(){
  if(isEmpty()){
    printf("Cannot remove element as queue is already empty.");
    return;
  }
  else if(front==rear){
    front=-1;
    rear=-1;
  }
  else{
    front=(front+1)%size;
  }
}

void main(){
  int choice=0,element, option=0;
  while(option!=4){
    printf("1. Input Restricted Queue\n2. Output Restricted
Queue\n3. No restriction queue\n4. Exit\nEnter your option: ");
    scanf("%d", &option);
    switch(option){
      case 1:
        while(choice!=7){
        printf("\n1. Insert Right\n2. Delete left\n3. Delete
right\n4. Is Empty?\n5. Is full?\n6. Display\n7. Exit");
        printf("\nEnter a choice: ");
        scanf("%d", &choice);
        switch(choice){
          case 1:
            printf("Enter an element: ");
            scanf("%d", &element);
            insert_right(element);
            break;
          case 2:
            delete_left();
            break;
          case 3:
            delete_right();
            break;
          case 4:
            if(isEmpty())
              printf("Queue is empty");
            else
              printf("Queue is not empty");
            break;
          case 5:
            if(isFull())
              printf("Queue is full");
            else
              printf("Queue is not full");
            break;
          case 6:
            display();
            break;
          case 7:
            exit(1);
            break;
          default:
            printf("Please enter a valid choice.");
            break;
        }
      }
    case 2:
      while(choice!=7){
        printf("\n1. Insert right\n2. Insert left\n3. Delete right\n4. Is Empty?\n5. Is full?\n6. Display\n7. Exit");
        printf("\nEnter a choice: ");
        scanf("%d", &choice);
        switch(choice){
          case 1:
            printf("Enter an element: ");
            scanf("%d", &element);
            insert_right(element);
            break;
          case 2:
            printf("Enter an element: ");
            scanf("%d", &element);
            insert_left(element);
            break;
          case 3:
            delete_right();
            break;
          case 4:
            if(isEmpty())
              printf("Queue is empty");
            else
              printf("Queue is not empty");
            break;
          case 5:
            if(isFull())
              printf("Queue is full");
            else
              printf("Queue is not full");
            break;
          case 6:
            display();
            break;
          case 7:
            exit(1);
            break;
          default:
            printf("Please enter a valid choice.");
            break;
        }
      }
      case 3:
        while(choice!=7){
        printf("\n1. Insert right\n2. Insert left\n3. Delete right\n4. Delete left\n5. Is Empty?\n6. Is full?\n7. Display\n8. Exit");
        printf("\nEnter a choice: ");
        scanf("%d", &choice);
        switch(choice){
          case 1:
            printf("Enter an element: ");
            scanf("%d", &element);
            insert_right(element);
            break;
          case 2:
            printf("Enter an element: ");
            scanf("%d", &element);
            insert_left(element);
            break;
          case 3:
            delete_right();
            break;
          case 4:
            delete_left();
            break;
          case 5:
            if(isEmpty())
              printf("Queue is empty");
            else
              printf("Queue is not empty");
            break;
          case 6:
            if(isFull())
              printf("Queue is full");
            else
              printf("Queue is not full");
            break;
          case 7:
            display();
            break;
          case 8:
            exit(1);
            break;
          default:
            printf("Please enter a valid choice.");
```

```
              break;
          }
          }
          case 4:
            exit(1);
            break;
      }
    }
  }
```

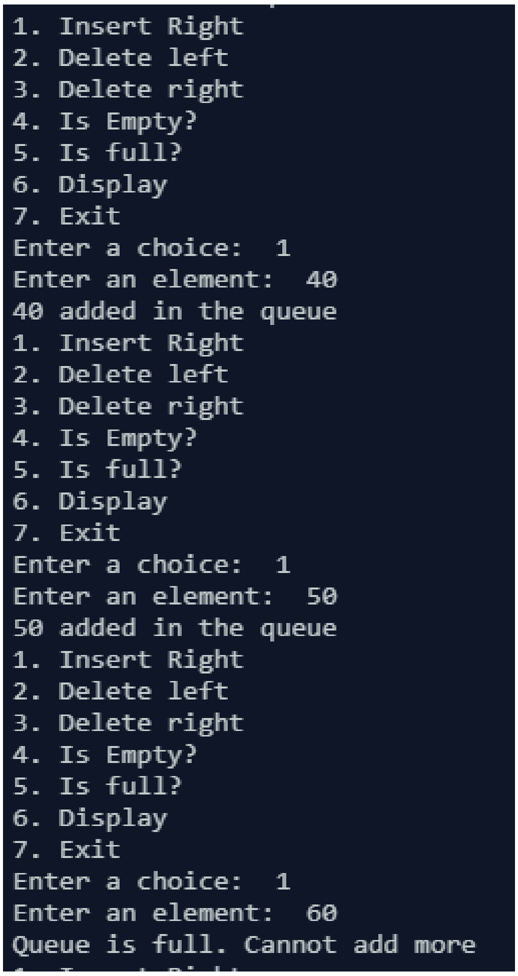
For verification, my code is available on:
https://repl.it/@ARGHYADEEPDAS/DSExpt4

OUTPUT SCREENS:

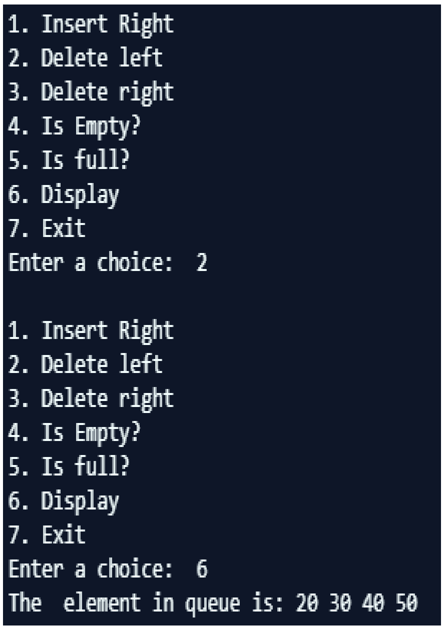
1. Input Restricted Queue:



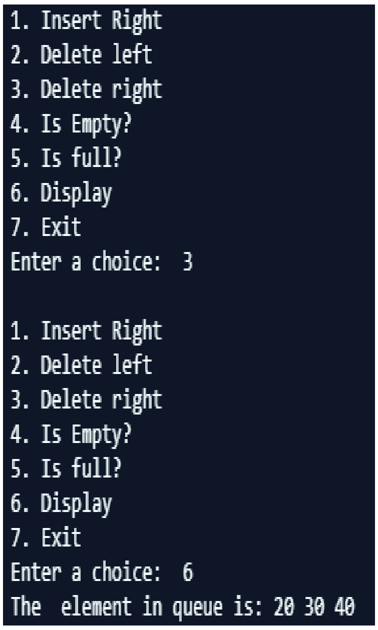
```
1. Input Restricted Queue
2. Output Restricted Queue
3. No restriction queue
4. Exit
Enter your option:  1

1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  10
10 added in the queue
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  20
20 added in the queue
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  30
```



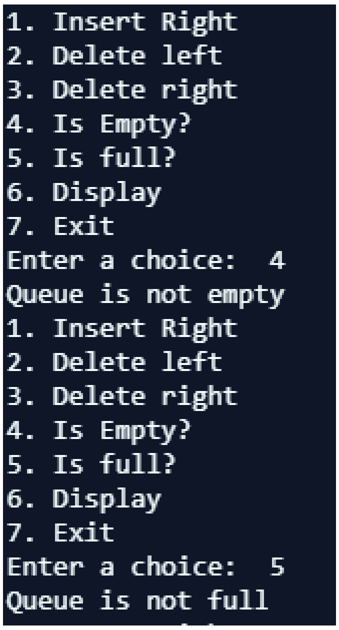
```
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  40
40 added in the queue
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  50
50 added in the queue
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  60
Queue is full. Cannot add more
```



```
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  2

1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  6
The  element in queue is: 20 30 40 50
```

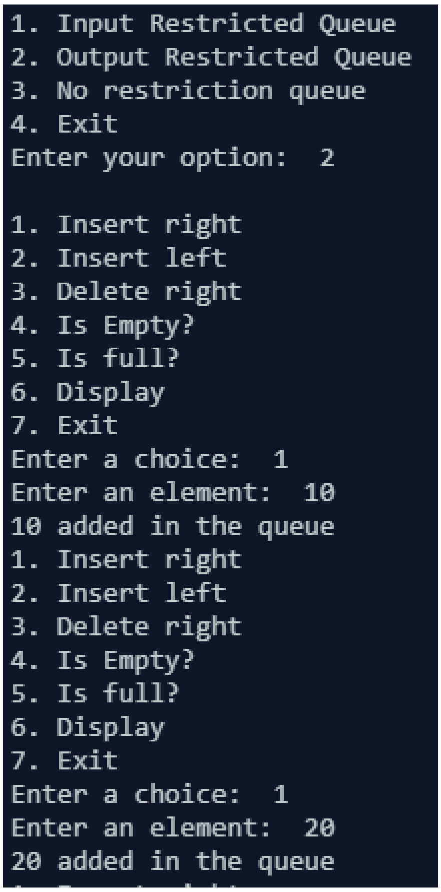


```
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  3

1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  6
The  element in queue is: 20 30 40
```

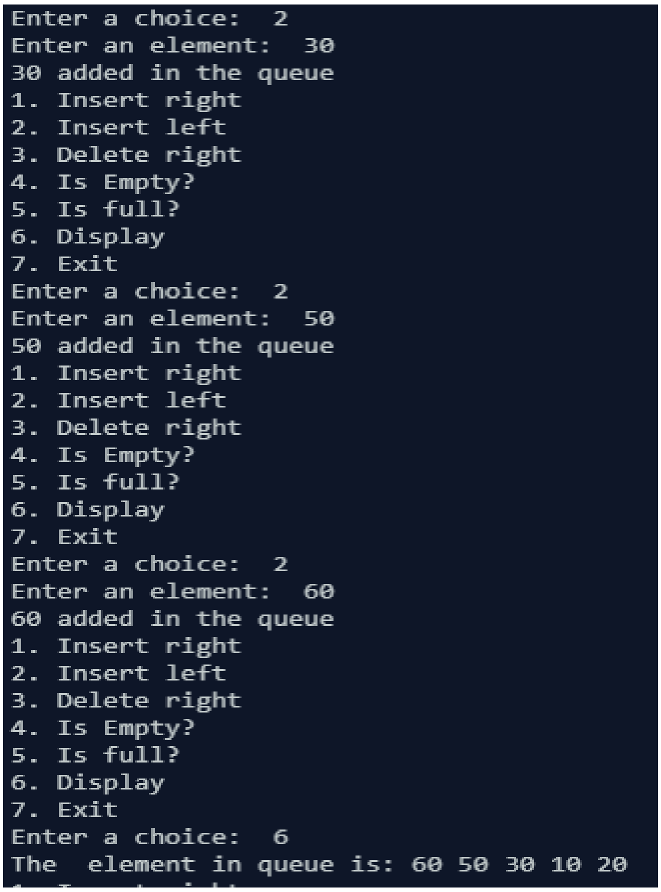


```
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  4
Queue is not empty
1. Insert Right
2. Delete left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  5
Queue is not full
```

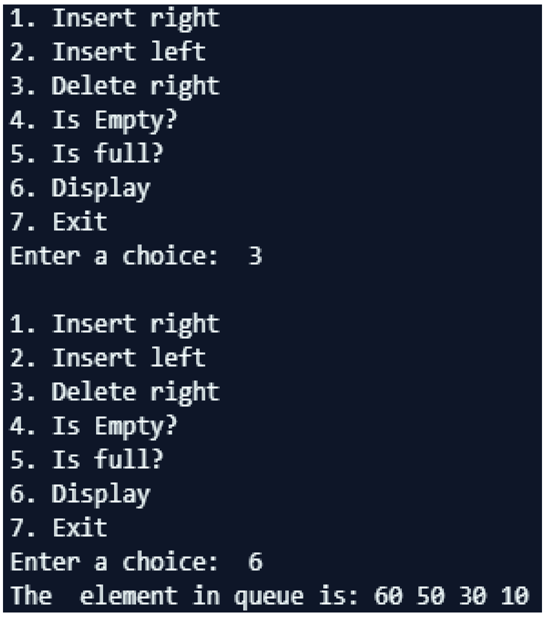
2. **Output Restricted Queue:**



```
1. Input Restricted Queue
2. Output Restricted Queue
3. No restriction queue
4. Exit
Enter your option:  2

1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  10
10 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  1
Enter an element:  20
20 added in the queue
```

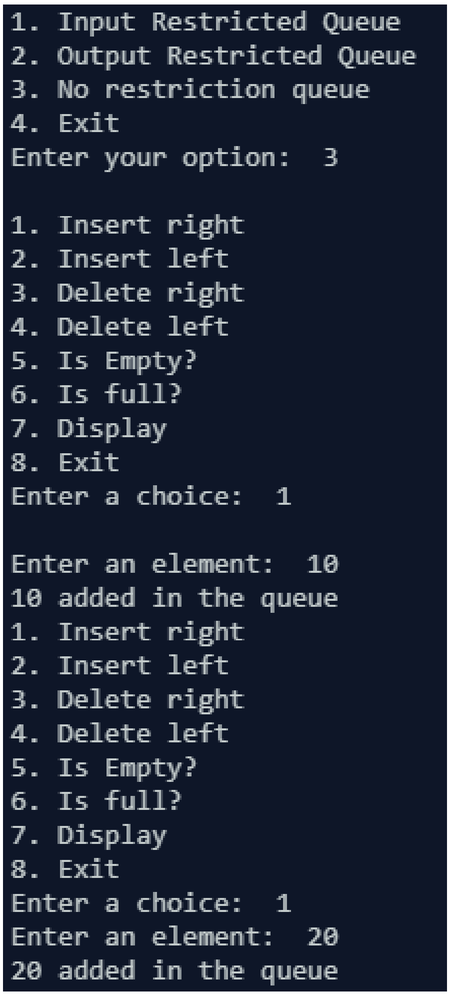


```
Enter a choice:  2
Enter an element:  30
30 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  2
Enter an element:  50
50 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  2
Enter an element:  60
60 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  6
The  element in queue is: 60 50 30 10 20
```

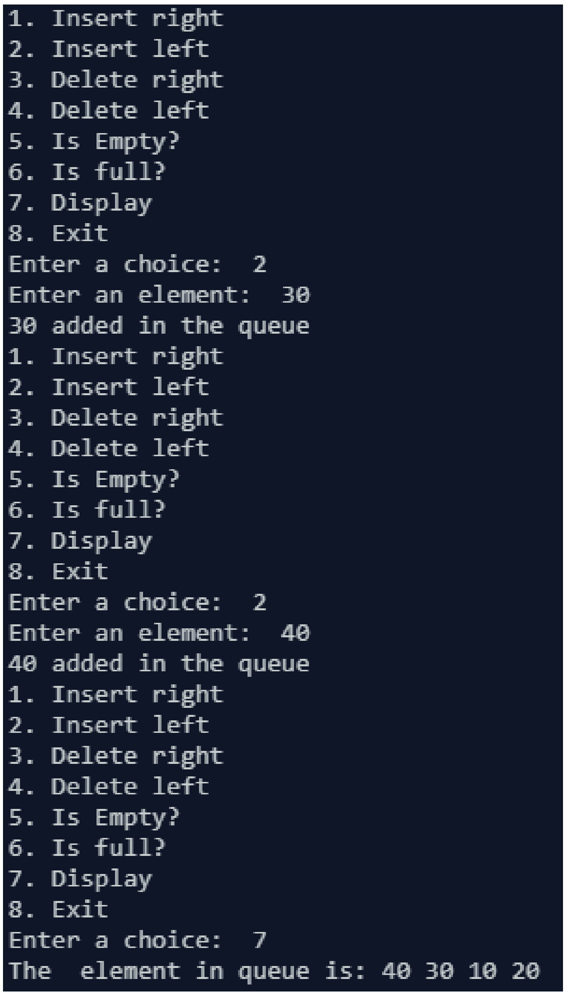


```
1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  3

1. Insert right
2. Insert left
3. Delete right
4. Is Empty?
5. Is full?
6. Display
7. Exit
Enter a choice:  6
The  element in queue is: 60 50 30 10
```

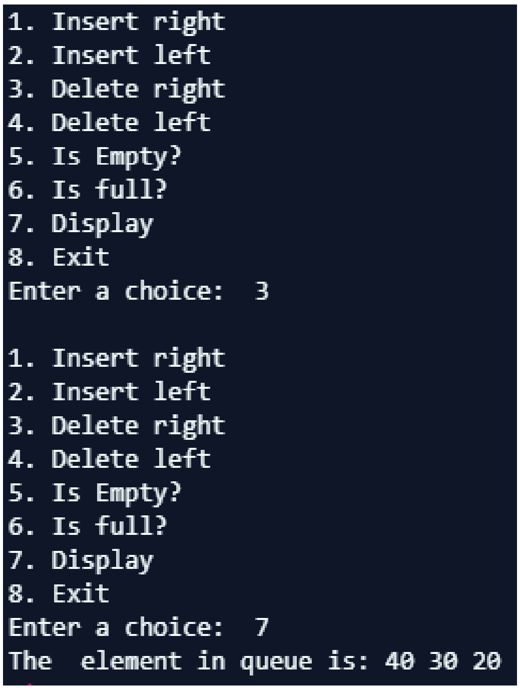
3. **No restriction queue:**



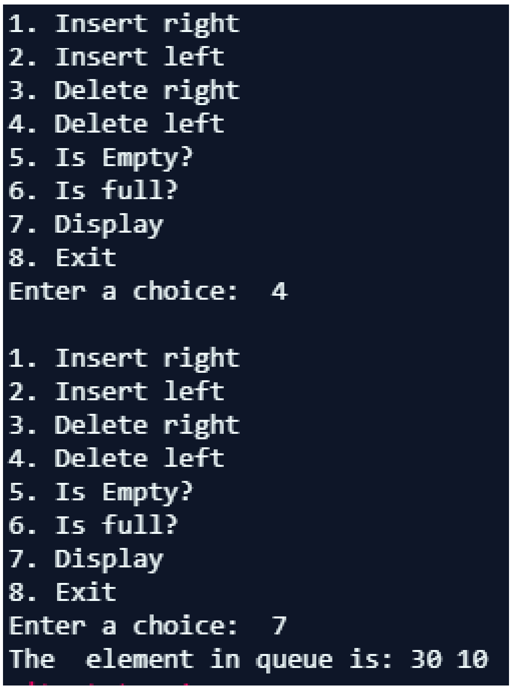
```
1. Input Restricted Queue
2. Output Restricted Queue
3. No restriction queue
4. Exit
Enter your option:  3

1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  1

Enter an element:  10
10 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  1
Enter an element:  20
20 added in the queue
```



```
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  2
Enter an element:  30
30 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  2
Enter an element:  40
40 added in the queue
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  7
The  element in queue is: 40 30 10 20
```



```
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  3

1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  7
The  element in queue is: 40 30 20
```



```
1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  4

1. Insert right
2. Insert left
3. Delete right
4. Delete left
5. Is Empty?
6. Is full?
7. Display
8. Exit
Enter a choice:  7
The  element in queue is: 30 10
```

CONCLUSION:

The program ran successfully as we were able to implement deque of all three types and the program gave correct output for various test cases and edge cases.