Experiment no. 5

→ Aim - A double-ended queue (deque) is a linear list in which additions & deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add & delete elements from either end of the deque.

Pre-requisite:

Knowledge of Queue
Types of Queue
Knowledge of double ended queue & different operations that can be performed on it.

Objective:
To simulate deque with function to add & delete elements from either end of the deque.

Input:
Size of array elements in the queue
Outcome:
Result of deque with funtions to add & delete elements from either end of the deque.

Theory:

Double-Ended Queue

A double-ended queue is an abstract data type similar to an ~~Simpilar~~ simple queue, it allows you to insert & delete from both sides means items can be added or deleted from the front or rear end.

Algorithm for Insertion at rear end

```
Step-1 : [Check for overflow] if (rear = = MAX) Print ("Queue is Overflow"); return;
Step-2 [Insert element] else
  rear = rear+1;
  q[rear]=no;
  [Set rear and Front pointer]

  If rear = 0
  rear; If front = 0
  Front ; Step-3 : return
```

Implemenation of Insertion at rear end.

```
Void add_items_rear()
 {
 int num;
printF("\n Enter Items to insert: "); scanF("%d", &num); If(rear = = MAX)
 {
  printF("\n Queue is OverFlow"); return;
 }
  else
{
 rear++; q[rear] = num; If (rear = = 0) rear =1; If (Front = = 0) front =1;
```

Algorithm for Insertion at front end

```
Step 1: [Check for the front position] if (front<=1)
Print ("Cannot add item at front end"); return;
Step2: [Insert at Front] else
front = front-1 ; q[Front]= no ; Step-3: Return
```

Implementation of Inseration at front end

```
Void add_item_front ()
{
 int num;
 printF ("\n Enter item to insert :"); scanF ("%d,"&num);

 iF(Front <=1)
 {
 printF("\n Cannot add item at front end");
 return;
 }
 else
 {
 Front--; q[Front] = num;
 }
}
```

Algorithm for Deletion from front end

```
Step-1 [Check for front pointer] if front =0
 print ("Queue is Underflow");
  return;
Step-2 [Perform deletion] else
  no = q[Front];
 print ("Deleted element is", no); [Set front and rear pointer]
 If front = rear front = 0; rear = 0;
 else front = front+1 ;
Step-3 :
   Return
```

Implementation of deletion from front end

```
void delete_item_Front ()
 {
  int num; if (front ==0)
  {
  printF("\n Queue is Underflow \n");
  return;
  }
  else
  {
   num = q [Front];
 printF ("\n Deleted items is %d \n", num); If (Front == rear)
  {
  Front =0; rear =0;
  }
  else
  {
  Front++;
  }
 }
 }
```

Algorithm for deletion from rear end

```
Step -1: [Check for the rear pointer]
 If rear =0
print ("Cannot delete value at rear end");
  return;
Step 2: [Perform deletion] else
 no= q(rear);
[Check for the front and rear pointer] If Front = rear
 front =0 ; rear =0; else
```

```
rear = rear-1;
print("Deleted element is", no);
Step3 - Return
```

Implementation of Deletion from rear end

```
Void delete_item_rear()
{
printF("\n Cannot delete item at rear end\n");
return;
}
else
{
num = q[rear]; If(Front == rear)
{
Front = 0; rear = 0;
}
else
{
rear --;
printF("\n Deleted items is %d\n", num):
}
}
}
```

Conclusion:

By this way, we can perform operations on double ended queue.