Queue

Queue is a restricted linear data structure in which insertion of data takes place at one end and deletion of data takes place in other end. The insertion end of the Queue is called the rear end. And the deletion end is called the front end. Queue is based upon FIFO List (First In First Out) list as the items inserted first is the first item to delete.

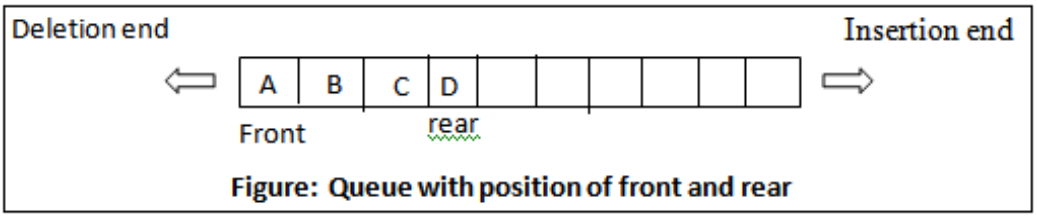

Figure: Queue with position of front and rear

Operations on a Queue

1. **Enqueue/Insert :** This is the operation that inserts an item in a queue.
2. **Dequeue / Delete:** This is the operation that removes an item from a queue.

State/ Error conditions in a Queue

1. **Overflow :** If a Queue is full and an attempt to insert an item in the Queue is made. This situation is called Overflow.
2. **UnderFlow :** If a Queue is empty and an attempt to delete an item from the Queue is made. This situation is called underflow.

Example: Consider an array implemented empty Queue of Size=5, show the status of the Queue stepwise after following operations are made
Insert(10), Insert(20),Insert(30), Delete, Insert(40), Insert(50), Delete, Delete, Insert(60),Delete, Delete

Assuming Array index starts with 1

Operations	QUEUE					Front , rear
						Front=0, Rear=0
Insert(10)	10					Front=1, Rear=1
Insert(20)	10	20				Front=1, Rear=2
Insert(30)	10	20	30			Front=1, Rear=3
Delete		20	30			Front=2, Rear=3
Insert(40)		20	30	40		Front=2, Rear=4
Insert(50)		20	30	40	50	Front=2, Rear=5

Delete			30	40	50		Front=3, Rear=5
Delete				40	50		Front=4, Rear=5
Insert(60)				40	50		OVERFLOW Front=4, Rear=5
Delete					50		Front=5, Rear=5
Delete							Front=0 Rear=0

Algorithm To insert an element in a Queue

QINSERT (QUEUE , MAX , front , rear, item)

//This Algorithm inserts an element item into a queue.

```
1. if rear =MAX,  then    // Is Queue Full
2.      Write "OVERFLOW" and Return.
   //End IF
3. if front =0 THEN
4.      front = rear = 1
5. else,rear = rear +1
  //End of If
7. QUEUE [rear] =item     // inserts new element
8: return.
```

Algorithm to delete an element in a queue.

QDELETE (QUEUE , MAX , front , rear)

//This algorithm deletes an element from a queue and assigns it to the value item

```
1. if front = 0  then     //Queue is empty
2.      write  "UNDERFLOW" and Return.
   //End of IF
3. item = QUEUE [front].
4. if front = rear, then
5.      front = rear =0
6. else, front = front+1.
   //End of IF
7. Return item.
```

Implementation of Queue using array in C

While implementing in C Language modify the start index of array to 0 and change the other code accordingly.

Program: Write a menu design program to implement following operations on a Array implemented Queue

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 10
void Qinsert(int);
void Qdelete();
void Qdisplay();
int QUEUE[MAX],front= -1,rear= -1;
int main()
     {
     int choice,no;
     char ch='Y';
     while(ch=='Y' || ch=='y')
          {
          printf("\n1:Insert");
          printf("\n2:Delete");
          printf("\n1:Display");
          printf("\nEnter a choice");
          scanf("%d",&choice);
          switch(choice)
              {
              case 1:
                   printf("\nEnter no to push");
                   scanf("%d",&no);
                   Qinsert(no);
                   break;
              case 2:
                   Qdelete();
                   break;
              case 3:
                   Qdisplay();
              }
          printf("\nContinue?(y/n)");     //fflush(stdin)
          ch=getchar();
          }
   }
```

```
void Qinsert(int no)
     {
     if(rear==MAX-1)
          {
          printf("Overflow");
          return;
          }
     if(front==-1)
          front=rear=0;
     else
          rear=rear+1;
     QUEUE[rear]=no;
     }

void Qdelete()
   {
   int no;
   if(front==-1)
       {
       printf("Underflow");
       return ;
       }
   no=QUEUE[front];
   if(front==rear)
       front=rear=-1;
   else
       front++;
   printf("%d is deleted",no);
   }

void Qdisplay()
   {
   int i;
   printf("\n");
   if(front==-1)
       {
       printf("\Underflow");
       return;
       }
   for(i=front; i<=rear;i++)
       {
       printf(" %d",QUEUE[i]);
       }
   }
```

CIRCULAR QUEUE

The problem of above-mentioned Queue is that even if the Queue is not full, it causes overflow if the REAR pointer is at the end of the Queue. To overcome this problem, Circular Queue is used.

A **Circular Queue** is "a Queue in which the element next to the last element is the first element." In this Queue, when REAR is at the MAX Position and FRONT is not at the 1st Location, an Item can be inserted at REAR after positioning REAR to the 1st Position.

It is similar to Array implemented General Queue with some extra condition to check overflow and to add or delete element from Queue. As in Circular Queue , if rear is at MAX and front is not in the beginning we can insert new items in rear by changing rear=rear +1, So there are two case for overflow condition.

1. Rear is just before front and (front=rear+1) and
2. Front is at beginning and rear is at Max position(Front=1 and Rear=Max)

Example: Consider an array implemented empty Queue of Size=5, show the status of the Queue stepwise after following operations are made.

Insert(10), Insert(20),Insert(30), Delete, Insert(40), Insert(50), Delete, Insert(60), Insert(70), Insert(80), Insert(90), Delete, Delete

Operations	QUEUE					Front , rear
						Front=0, Rear=0
Insert(10)	10					Front=1, Rear=1
Insert(20)	10	20				Front=1, Rear=2
Insert(30)	10	20	30			Front=1, Rear=3
Delete		20	30			Front=2, Rear=3
Insert(40)		20	30	40		Front=2, Rear=4
Insert(50)		20	30	40	50	Front=2, Rear=5
Delete			30	40	50	Front=3, Rear=5
Insert(60)	60			40	50	Front=4, Rear=1
Insert(70)	60	70		40	50	Front=4, Rear=2
Insert(80)	60	70	80	40	50	Front=4, Rear=3
Insert(90)	60	70	80	40	50	OVERFLOW Front=4 Rear=3
Delete	60	70	80		50	Front=5 Rear=3
Delete	60	70	80			Front=1 Rear=3

Algorithm to insert an element in a circular Queue

CQInsert (CQUEUE , MAX , front , rear , item)
// This algorithm inserts an element item into a circular queue CQUEUE of size MAX.

```
1. if front = 1 and  rear =MAX  or if front =rear +1 then
2.      Write "OVERFLOW" and return
3. if front = 0 then           //Queue initially empty
4.      front = 1 and rear =1.
   else  if rear = MAX then
5.      rear = 1.
6. else,rear = rear +1.
   //End IF
7. CQUEUE [rear] =item
8. return.
```

Algorithm to delete an element from a circular Queue

CQDelete(CQUEUE, MAX, front, rear)

//This algorithm deletes an item from a circular queue

```
1. IF front = 0  THEN          //Queue is empty
2.      Write "UNDERFLOW" and Return.
   //End of IF
3. item = CQUEUE[front].
4. IF front = rear, THEN
5.      front = rear =0
6. ELSE IF front = MAX
7.      front=1
8. ELSE, front = front+1.
   //End of IF
9. Return item.
```

Program:

Write a program to implement all the operations of a Circular Queue using an array

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
void CQinsert(int);
void CQdelete();   void display();
int QUEUE[MAX],front=-1,rear=-1;
int  main()    {
     int choice,no;
     char ch = 'Y';
     while(ch == 'y' || ch == 'Y')
          {
          printf("\n1:Insert");
          printf("\n2:Delete");
          printf("\n1:Display");
          printf("\nEnter a choice");
          scanf("%d",&choice);
          switch(choice)      {
          case 1:   printf("\nEnter no to push");
               scanf("%d",&no);
               CQinsert(no);
               break;
          case 2:   CQdelete();
               break;
          case 3:   CQdisplay();
               }
          printf("\nContinue?(y/n)");
          getchar();  ch=getchar();
          }
     }
void CQinsert(int no)
     {
     if((front==0 &&rear==MAX-1) || front==rear+1)
          {
          printf("\nOverflow");
          return;
          }
     if(front==-1)
          front=rear=0;
     else if (rear==MAX-1)
          rear=0;
     else
          rear=rear+1;
     QUEUE[rear]=no;
     }
```

```
void CQdisplay()
     {
     int i;
     printf("\n");
     if(front==-1)
          {
          printf("\nNo item in Queue");
          return;
          }
     if(front<=rear)
          {
          for(i=front; i<=rear;i++)
               printf(" %d",QUEUE[i]);
          }
     else
          {
          for(i=front; i<=MAX-1;i++)
               printf(" %d",QUEUE[i]);
          for(i=0;i<=rear;i++)
               printf(" %d",QUEUE[i]);
          }
     }

void CQdelete()
     {
     int no;
     if(front==-1)
          {
          printf("\nUnderflow");
          return ;
          }
     no=QUEUE[front];
     if(front==rear)
               front=rear=-1;
     else if (front==MAX-1)
               front=0;
     else
               front++;
     printf("\n%d is deleted",no);
     }
```

DOUBLE ENDED QUEUES

A DEQUE is a linear list in which elements can be added or removed either end but not in the middle. It can be viewed as a Queue as well as stack, where an item can be inserted and deleted in both Front end as well as in rear end.

Operations on a DEQUEUE

1. Insertion at Rear(Right)
2. Insertion at Front(Left)
3. Deletion at Front(Left)
4. Deletion at Rear(Right)

There are two types of DEQUE.

1. **Input Restricted DEQUE** is a DEQUE, which allows insertion of item at only one end of the Queue i, e in rear end but allows deletions at both end of the Queue. Following operations are allowed on this type of Queue.
    i. Insertion at Front(Left)
    ii. Deletion at Front(Left)
    iii. Deletion at Rear(Right)
2. **Output Restricted DEQUE** is a DEQUE which allows deletions at only one end of the of the queue i.e. the Front end and but allows insertions at both the end of the Queue. Following operations are allowed on this type of Queue.
    i. Insertion at Rear(Right)
    ii. Deletion at Front(Left)
    iii. Deletion at Rear(Right)

Insertion at rear: The procedure is same as the algorithm for insertion of the item in a circular queue. Refer to circular queue insertion algorithm.

Deletion at front: The procedure is same as the algorithm for deletion of the item in a circular queue. Refer to circular queue deletion algorithm.

Deletion at Rear: This operation deletes and returns the item from the rear-end of DEQUEUE. Normally, when this operation is called, rear is set to (rear – 1). But when the rear is at beginning position, deletion will set rear to the MAX position of the DEQUEUE.

Insertion at Front : This operation inserts the item at the front end of DEQUEUE. Normally when this operation is called Front is set to (Front -1). However, when the Front is at beginning position, front is to be set to MAX. Then item can be inserted at front position of DEQUEUE.

```
DEQDeleteAtRear (DQUEUE, MAX, front, rear)
```

//This algorithm deletes an item from a double ended queue from rear end

```
1. IF front = 0  THEN            //Queue is empty
2.      Write "UNDERFLOW" and Return.
   //End of IF
3. item = DQUEUE[rear].
4. IF front = rear, THEN
5.      front = rear =0
6. ELSE IF rear=1
7.      Rear=MAX
8. ELSE, rear = rear - 1
   //End of IF
9. Return item.
```

```
DEQInsertAtFront (DQUEUE , MAX , front , rear , item)
```

// This algorithm inserts an element item at Front end of DEQUEUE of size MAX .

```
1. IF front = 1 and  rear =MAX  OR IF front =rear +1 THEN
2.      Write "OVERFLOW" and RETURN
   //END IF
3. IF front = 0 THEN           //Queue initially empty
4.      front = 1 and rear =1.
5. ELSE  IF front = 1 THEN
6.      front = MAX
7. ELSE
8.      front = front - 1.
   //End IF
9. DQUEUE [front] = item
10.     Return.
```