Exercise 3 Queue

3. Write a C++ program for implementation of Queue Using
a) Array b) Linked List

Objective: The objective of this exercise is to enable you to program a queue and to perform operations on it.

Procedure and description:
A queue is a linear list of elements in which deletions can take place only at one end, called the front and insertions can take place only at the other end, called the rear.

a) Implementation of Queue Using Array
1. Insert: For inserting an element into the queue with the procedure QINSERT(), before insertion need to check the overflow status of the queue and find the current position of REAR and increment that with one and this is the new location for inserting a new item .

Notations Used:

QUEUE	:	An Array containing the elements of the Queue
FRONT	:	A pointer pointing to location of queue where deletion can be done.
REAR	:	A pointer pointing to Current location of queue where insertion can be done.
ITEM	:	New data to be added to the list
N	:	Size of Queue (maximum number of element that can be held by queue)

Algorithm:

```
QINSERT (QUEUE, N, FRONT, REAR, ITEM)
Step 1: If FRONT=1 and REAR =N, or FRONT=REAR+1 then
            Write OVERFLOW, and Return

Step 2: [Find new value of REAR]
            If FRONT: =NULL, then: [Queue is empty initially]
               Set FRONT: =1 and REAR: =1
               Else if REAR=N then:
                     Set REAR: =1.
```

```
        Else
        Set REAR: =REAR+1
Step 3: Set QUEUE [REAR]:=ITEM
Return
```

2. Deletion: For deletion of an item from the queue with the procedure QDELETE() which checks for the underflow status if queue contains items it deletes an first element from the queue by assigning it to the variable ITEM and calculate new value for REAR.

Algorithm:

```
QDELETE (QUEUE, N, FRONT, REAR, ITEM)
Step 1: If FRONT: =NULL, then write: UNDERFLOW and Return.
Step 2: Set ITEM: =QUEUE [FRONT].
Step 3: If FRONT = REAR, then:          // only one element.
        Set FRONT: =NULL and REAR: =NULL.
          Else if FRONT: = N then:
        Set FRONT: =1.
       Else:
        Set FRONT: = FRONT +1.
Return.
```

Expected Output:

After executing program. Enter Input choice either insertion or deletion.

For better understanding see below pictorial representation. Insertion can be performed at rear end and deletion can be performed at front end.

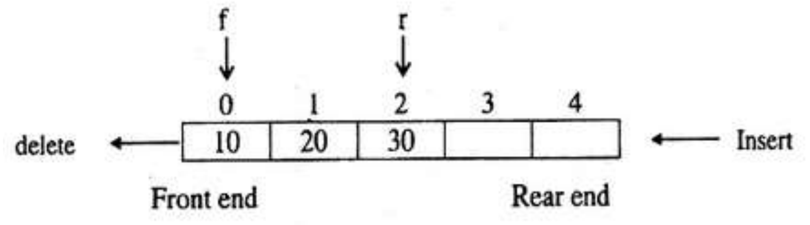

Fig.: Initial Queue using array

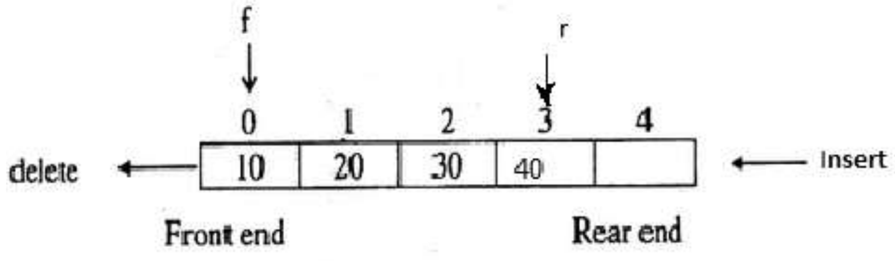

Fig.: After inserting element 40 at rear end

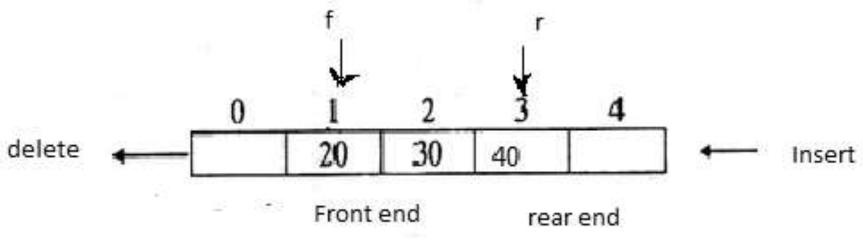

Fig.: After deleting element 10 at front end

b) Implementation of Queue using LINKED LIST

Procedure and description:

Here we are going to implement queue in linked list. Queues are very much like linked list except the ability to manipulate items on the lists, a linked queue is a queue implemented as a linked list with two pointer variables FRONT and REAR pointing to the nodes which is in the FRONT and REAR of the queue.

1. Insertion of element in linked queue

In linked queue representation while inserting an element a new node will be availed from the AVAIL list which holds the ITEM, will be inserted as the last node of the linked list representing queue. The rear point will be updated in order to point the node which is recently entered.

Algorithm:

Notations Used:

FRONT : A pointer pointing to location of queue where deletion can be done.

REAR	:	A pointer pointing to Current location of queue where insertion can be done.
DATA	:	Node First part called DATA, contains data value
LINK	:	Node Second part called LINK, contains address
START	:	The address of the first node
AVAIL	:	Address of the pointer in the linked list
ITEM	:	New data (Node) to be added to the list
LOC	:	Address, where the New data (Node) is to be inserted

```
LINKQ_INSERT (INFO, LINK, FRONT, REAR, AVAIL, ITEM)
Step 1: If AVAIL = NULL, then write OVERFLOW and Exit
Step 2: Set NEW: =AVAIL and AVAIL: =LINK [AVAIL]
            [Remove first node from AVAIL list]
Step 3: Set INFO [NEW]:=ITEM and LINK [NEW] =NULL
            [Copies ITEM to new node]
Step 4: If (FRONT=NULL) then FRONT=REAR=NEW
         [If queue is empty then ITEM is the first
            element in the queue Q]
          Else Set LINK [REAR]:=NEW and REAR = NEW
          [REAR points to the new node appended to the end of list]
Exit.
```

2. Deletion of element from linked queue

The deletion can happen only from the FRONT end. In case of deletion the first node of the list pointed to by FRONT is deleted and the FRONT pointer is updated to point to the next node in the list and the deleted node will be return to the AVAIL list.

```
LINKQ_DELETE (INFO, LINK, FRONT, REAR, AVAIL, ITEM)
Step 1: If (FRONT=NULL) then Write: UNDERFLOW and Exit
Step 2: Set TEMP=FRONT [if queue is not empty]
Step 3: ITEM=INFO (TEMP)
Step 4: FRONT=LINK (TEMP) [Reset FRONT to next element]
Step 5: LINK (TEMP) =AVAIL and AVAIL=TEMP
          [Return deleted node to AVAIL list]
 Exit.
```

Expected Output:

After executing program. Enter INPUT things

For better understanding see below pictorial representation. Insertion can be performed at rear end and deletion can be performed at front end.

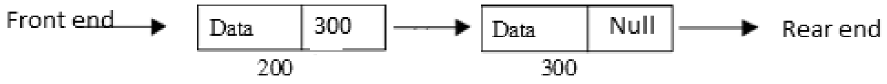

Fig.: Initial Queue using linked list

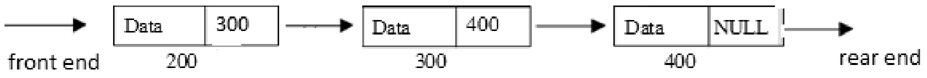

Fig.: After inserting node at rear end

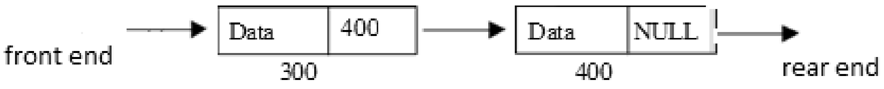

Fig.: After deletion of node at front end