QUEUE

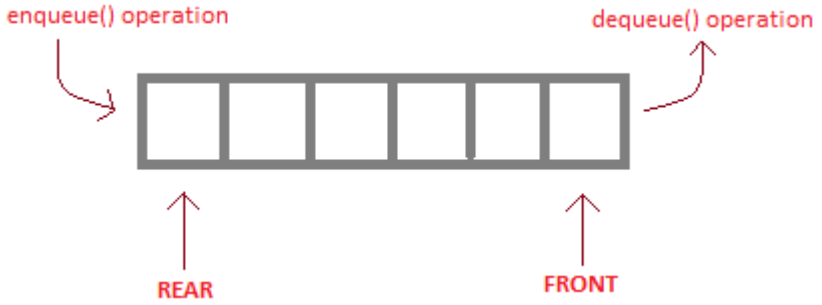
Presented by : Dr. Aparna Kumari

Introduction

- It is an ordered list of elements of similar data types which permits deletions to be performed at one end of a list and insertions at the other.
- The information in such a list is processed in the same order as it was received, that is, on a first-in, first-out (FIFO) or first-come, first-serve (FCFS) basis.
- From the "front" or "head" end, one can delete the items.
- From the "rear" or "tail" end, one can insert the items.
- The typical example can be a queue of people who are waiting for a ticket at ticket counter. Any new person is joining at one end of the queue, you can call it as the rear end. When the chance arrives the person at the other end first buys the ticket, you can call it as the front end of the queue.

Introduction

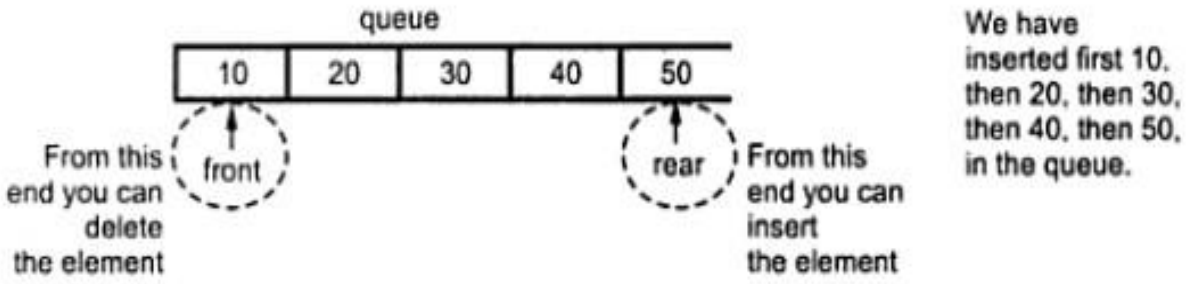
- The process to add an element into queue is called “Enqueue” and the process of removal of an element from the queue is called “Dequeue”.



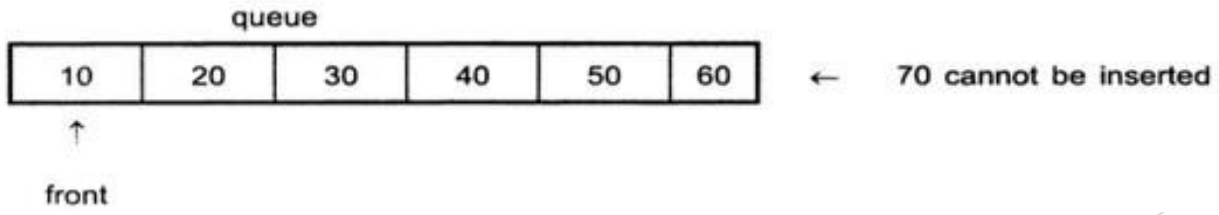
- Various operations on the queue are:
  - Queue overflow
  - Insertion of the element into the queue
  - Queue underflow
  - Deletion of the element from the queue
  - Display the queue

Insertion of element into the queue

- The insertion of any element in the queue will always take place from the rear end.

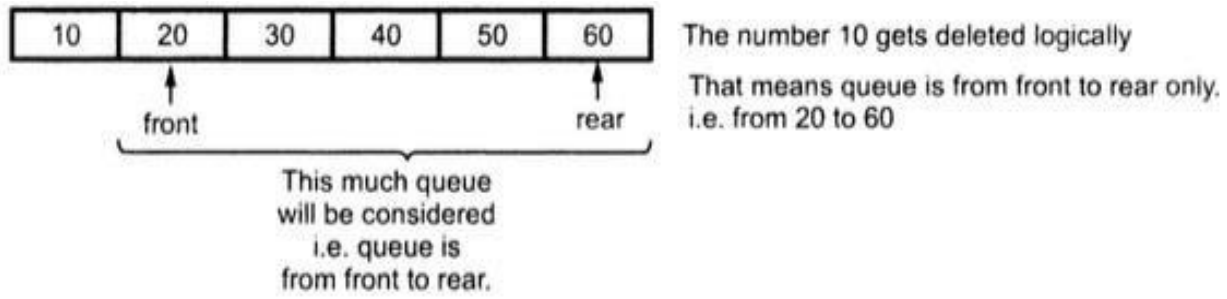


- Before performing insert operation, you must check whether the queue is full or not. If the rear pointer is going beyond the maximum size of the queue, then the queue overflow occurs.

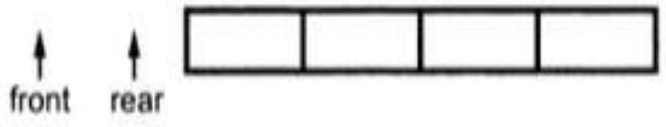


Deletion of element from the queue

- The deletion of any element in the queue will always take place by the front end always.

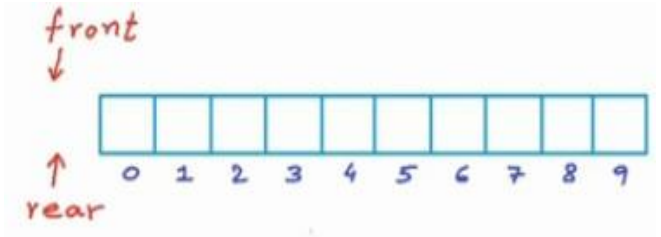


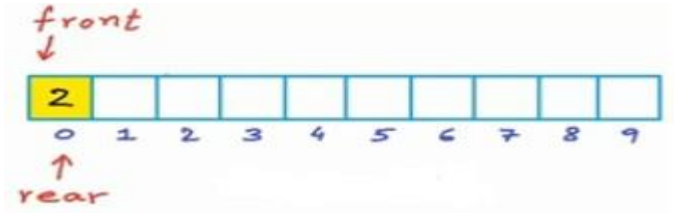
- Before performing delete operation, you must check whether the queue is empty or not. If the queue is empty, you can not perform the deletion. The result of illegal attempt to delete an element from the empty queue is called the queue underflow condition.

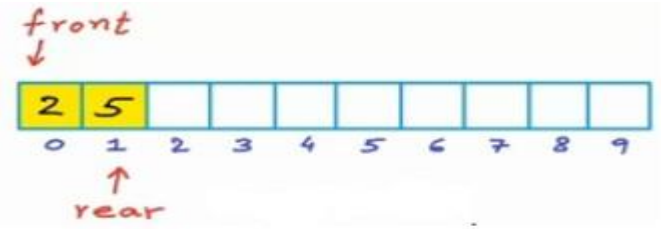


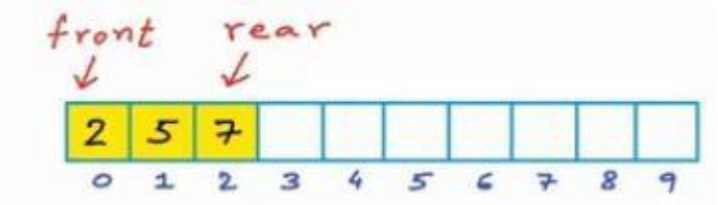
Tracing of Queue

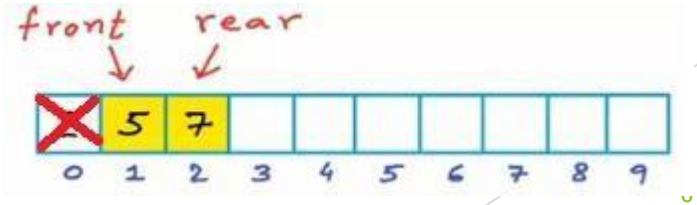
- Initially queue is empty, with F = R = -1
- Enqueue (2)
  - R = 0
  - Q[0] = 2
  - F = 0
- Enqueue (5)
  - R = 1
  - Q[1] = 5
- Enqueue (7)
  - R = 2
  - Q[2] = 7
- Dequeue( )
  - Y = Q[0]
  - F = 1



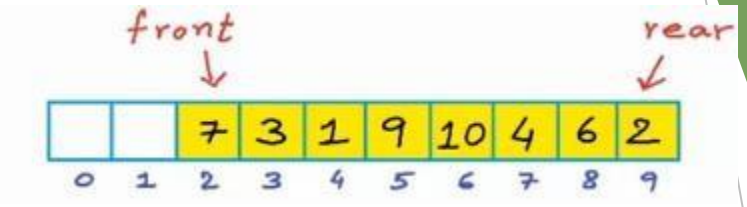




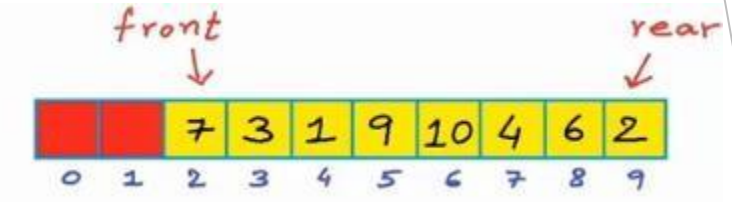




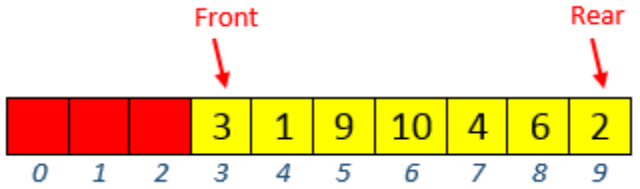
Tracing of Queue



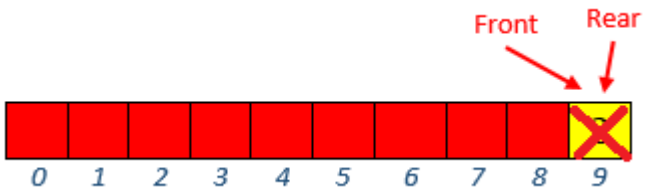
- After some Enqueue and Dequeue operations
  - Queue is now full as R = N - 1, cannot add more elements
- Wastage of two cells as they will never be used.

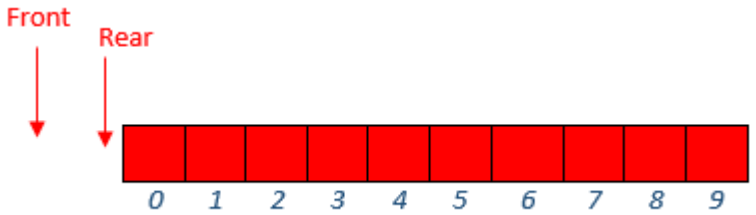


- Dequeue( )
  - Y = Q[2]
  - F = 3



- After some Dequeue operations, when only element is left in the queue, F = R
  - F = R = -1





Tracing of Queue

```
void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

Queue Algorithm

- Procedure QINSERT(Q, F , R, N, Y). Given F and R pointers to the front and rear elements of the queue, a queue Q consisting of N elements and an element Y, this procedure inserts Y at the rear of the queue. Prior to the first invocation of the procedure, F and R have been set to -1.

1. [Overflow?]
   If R ≥ N - 1
   then Write('OVERFLOW')
   Return

2. [Increment rear pointer]
   R ← R + 1

3. [Insert element]
   Q[R] ← Y

4. [Is front pointer properly set?]
   If F = -1
   then F ← 0
   Return

Queue Algorithm (Continue…)

```
void QINSERT()
{
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Insert the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
}
```

Queue Algorithm

- Function QDELETE(Q, F , R). Given F and R pointers to the front and rear elements of the queue, respectively, and the queue Q to which they correspond, this function deletes and returns the last element of the queue. Y is a temporary variable.

1. [Underflow?]
   If F = -1
   then Write('UNDERFLOW')
   Return (0)

2. [Delete element]
   Y ← Q[F]

3. [Queue empty?]
   If F = R
   then F ← R ← -1
   else F ← F + 1

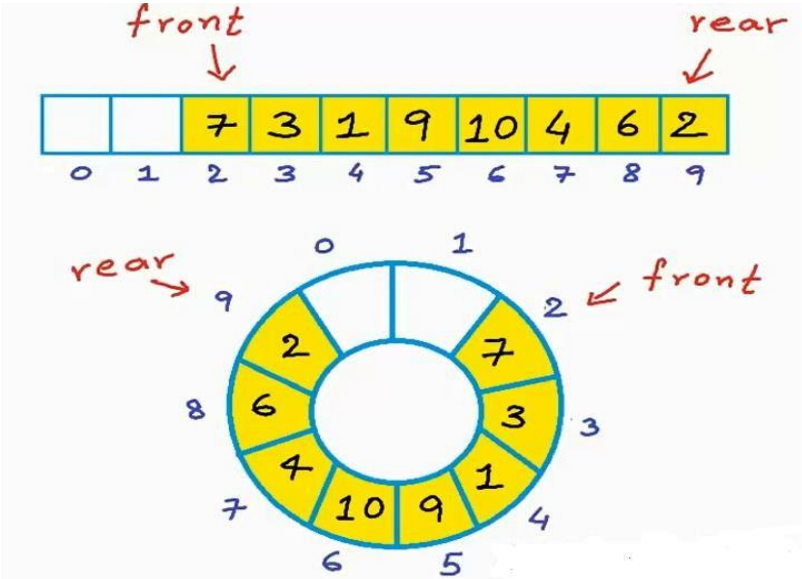
4. [Return element]
   Return (Y)

Queue Algorithm (Continue…)

```
void QDELETE()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
}
```

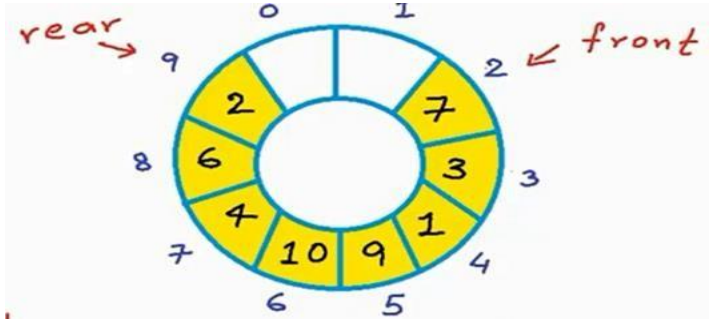
Circular Queue

- Wastage of space in linear queue
- Items cannot be added when overflow condition takes place, even if there is space because of deletion of items.
- Main advantage of circular queue is utilize the space of the queue fully.
- In Circular queue, as we traverse an array, we can imagine that there is no end in the array, from 0 we can go to 1, from 1 we can go to 2, and finally when we reach the last index in the array, like in this case, its 9, the next index for me should be 0.

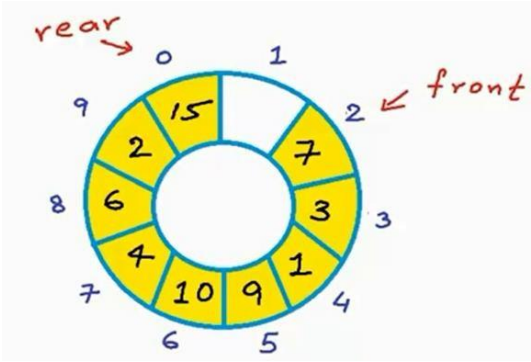


Tracing of Circular Queue

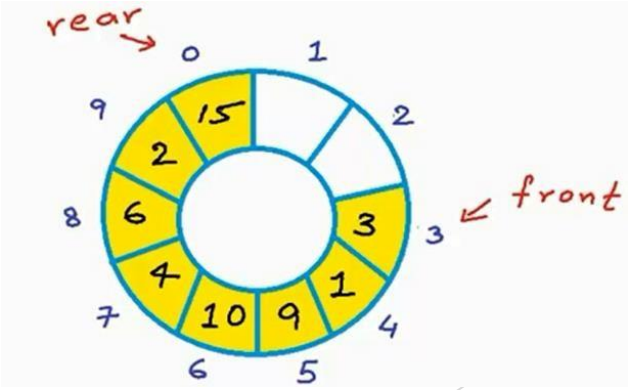
- Initial queue condition →



- Enqueue(15)
  - Since, rear = 9 = N – 1 (9) → rear = 0
  - Q[0] = 15



- Dequeue( )
  - Y = Q[2]
  - Since, front = 2 ≠ N – 1 = 9, hence front = 2 + 1 = 3



Circular Queue Algorithm

- Procedure CQINSERT (F , R, Q, N, Y). Given the pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, and an element Y, this procedure inserts Y at the rear of the queue. Initially, F and R are set to -1.

1. [Reset rear pointer?]
   If R = N -1
   then R ← 0
   else R ← R + 1

2. [Overflow?]
   If F = R
   then Write('OVERFLOW')
   Return

3. [Insert element]
   Q[R] ← Y

4. [Is front pointer properly set?]
   If F = -1
   then F ← 0
   Return

Circular Queue Algorithm

- Function CQDELETE (F , R, Q, N). Given the pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, this function deletes and returns the last element of the queue. Y is a temporary variable.

1. [Underflow?]
   If F = -1
   then Write('UNDERFLOW')
        Return(0)

2. [Delete element]
   Y ← Q[F]

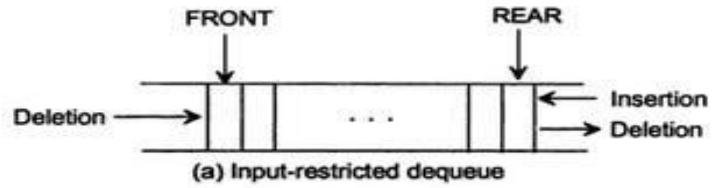
3. [Queue empty?]
   If F = R
   then F ← R ← -1
        Return(0)

4. [Increment front pointer]
   If F = N - 1
   then F ← 0
   else F ← F + 1
        Return(Y)

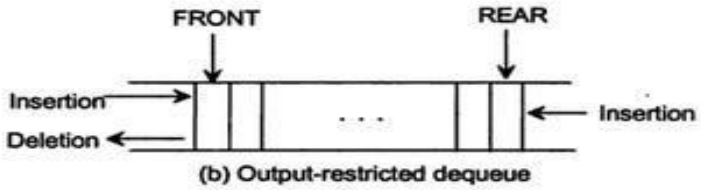
Double Ended Queue (Deque)

- A deque is a list in which the elements can be inserted or deleted at either end.
- Elements can be added or removed from either the front (head) or the rear (tail) end.
- However, no element can be added and deleted from the middle.
- Is implemented as circular queue.
- There are two types of deque:
  - Input restricted deque: In this deque, insertions can be done only at one of the ends, while deletions can be done from both ends.
  - Output restricted deque: In this deque, deletions can be done only at one of the ends, while insertions can be done from both ends.

A deque structure.



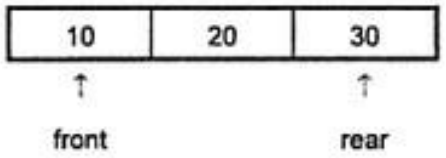
(a) Input-restricted dequeue



(b) Output-restricted dequeue

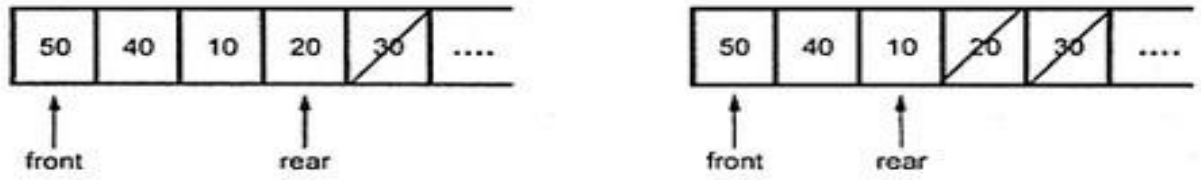
Double Ended Queue (Deque)

- Normally, we insert the elements by the rear end and delete the elements from front end. Let us say we inserted the elements 10, 20, 30 by rear end.



- Now, if we wish to insert any element from front end, then first we have to shift all the elements to the right.
- For example, if we want to insert 40 by front end, then the deque will be

(a) Insertion by front end



(b) Deletion by rear end

Double Ended Queue (Deque)

- Input Restricted Queue
  - Insert from rear (same as previously discussed)
  - Delete from front (same as previously discussed)
  - Delete from rear
- Output Restricted Queue
  - Insert from rear (same as previously discussed)
  - Insert from front
  - Delete from front (same as previously discussed)

Deque Algorithm

- Procedure DQ_INSERT_FRONT (F , R, Q, N, Y). Given the pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, and an element Y, this procedure inserts Y at the front of the queue. Initially, F and R are set to -1.

1. [Overflow?]
   If F = R + 1
   then Write('OVERFLOW')
   Return

2. [Is front and rear properly set?]
   If F = -1
   then F ← R ← 0

3. [Set front pointer]
   If F = 0
   then F = N - 1
   else F = F - 1

4. [Insert element]
   Q[F] = Y

Deque Algorithm

- Function DQ_DELETE_REAR (F , R, Q, N). Given the pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, this function deletes and returns the element from rear of the queue. Y is a temporary variable.

1. [Underflow?]
   If F = -1
   then Write('UNDERFLOW')
        Return(0)

2. [Delete element]
   Y ← Q[R]

3. [Queue empty?]
   If F = R
   then F ← R ← -1
        Return(0)

4. [Increment rear pointer]
   If R = 0
   then R ← N - 1
   else R ← R - 1
        Return(Y)

Thankyou

ANY QUESTIONS??