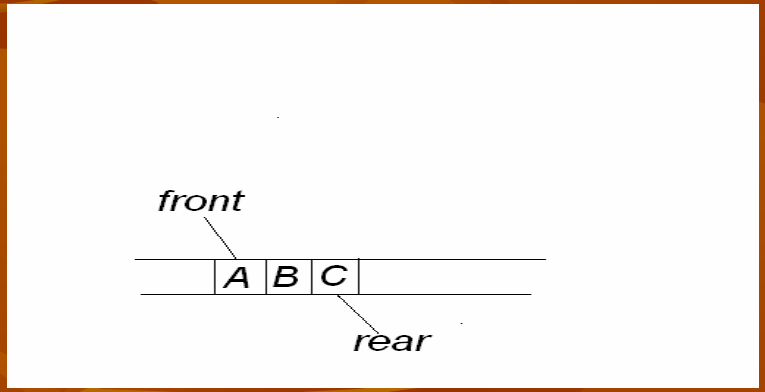
Queues

- Sequential representation:-

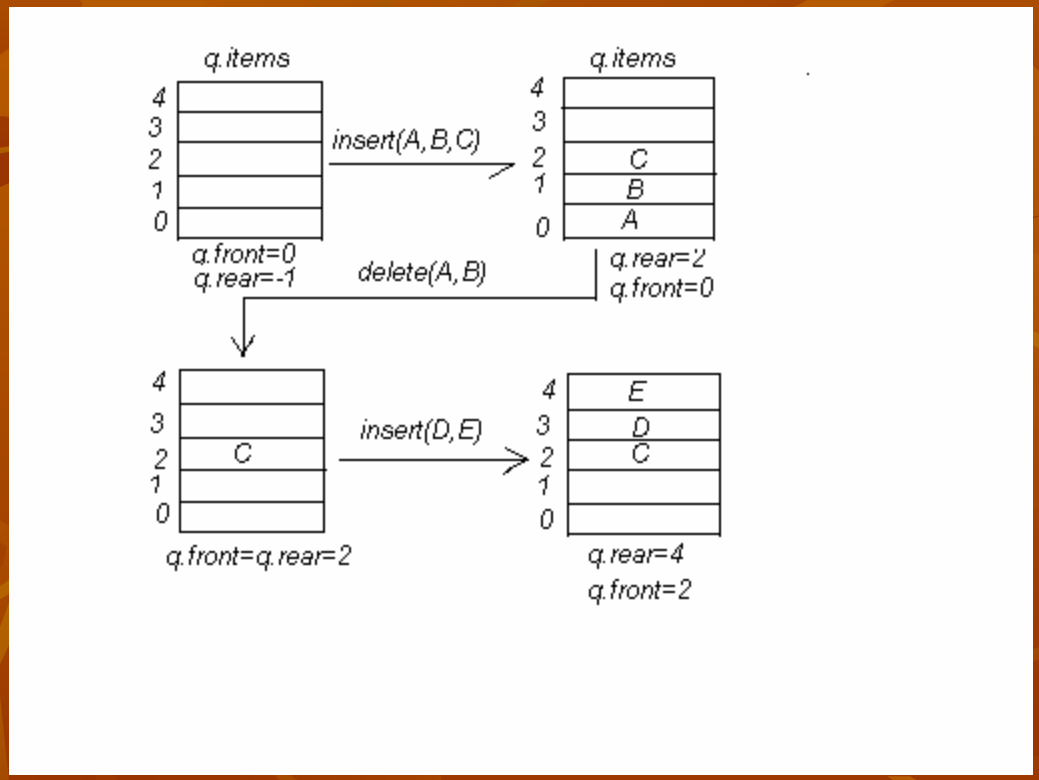


- Insertion takes place at rear end
- Deletion takes place at front end

- C- implementation
- #define MAXQUEUE 100
- struct queue{

```
int items[MAXQUEUE] ;
int front, rear ;
}q ;
```

- Initially :
- q.front = 0 ; q.rear = -1 ;
- insert(q, x) : q.items[++q.rear]=x ;
- x=remove(q) : x=q.items[q.front++] ;


q.items
4
3
2
1
0
q.front=0
q.rear=-1
insert(A,B,C)
q.items
C
B
A
q.rear=2
q.front=0
delete(A,B)
C
q.front=q.rear=2
insert(D,E)
E
D
C
q.rear=4
q.front=2

- **Queue is empty**
- q.rear < q.front
- #elements in the queue at ant time =
- q.rear – q.front + 1

- **Disadvantage**
- Though two elements of the array are empty no further insertion provide since q.rear = MAXQUEUE -1

- **Solution**
- **1)** x = remove(q) is modified as follows :

- x = q.items[0] ;
- for (i=0; i < q.rear ; i++)
- q.items[i] = q.items[i+1] ;
- q.rear-- ;

- The queue no longer contain a front field, since the element at position 0 of the array is always at the front of the queue.

- Empty queue is equivalent to :

  q.rear = -1

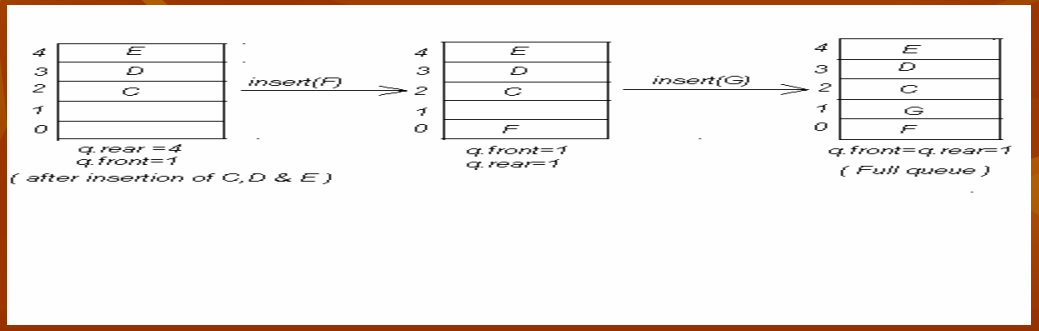
2)Use Circular Array representation:

- If q.rear = MAXQUEUE -1 and we want to insert reset q.rear←0

- #define MAXQUEUE 100
- struct queue{
- int items[MAXQUEUE]
- int front, rear ;
- };
- struct queue q ;
- q.front = q.rear = MAXQUEUE - 1

- int empty (struct queue *pq)
- {
- return (pq->front = pq->rear) ? TRUE
  : FALSE
- }/*end empty*/

- **Ambiguity between full & empty queue :**



- To distinguish between full and empty queue we sacrifice one element of the array and allow a queue to grow only as large as one less than the size of an array.
- int remove (struct queue *pq)

```
{   if empty( pq ){
        printf ("queue underflow");
        exit(1) ;
    }/*end if*/
```

```
    if (pq->front == MAXQUEUE-1 )
            pq->front = 0 ;
    else   (pq->front)++
   return ( pq->items[pq->front] ) ;
}/*end remove*/
```

pq->front is always pointing to an empty location in the array

```
Void insert ( struct queue *pq, int x )
{  /*make room for new element */
    if ( pq->rear = MAXQUEUE -1 )
                pq->rear = 0 ;
    else    (pq->rear)++ ;
    /* check for overflow */
    if ( pq->rear == pq->front ){
        printf("queue overflow") ;
        exit(1) ;
    }/*end if */
    pq->items[pq->rear] = x ;
    return ;
} /*end insert */
```