DATA STRUCTURES (ITPC-203)

Queues

Mrs. Sanga G. Chaki

Department of Information Technology

Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

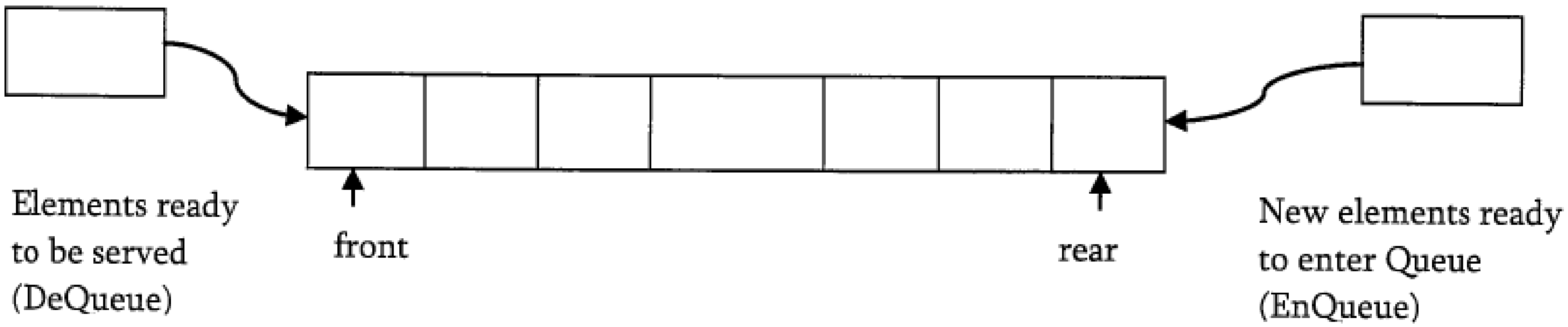
Contents

- Queues,
- Operations on Queue: Create, Add, Delete,
- Full and Empty,
- Circular queues,
- Array and linked implementation of queues in C,
- Deque
- Priority Queue

What is a Queue?

1. A queue is an ordered list in which insertions are done at one end (rear) and deletions are done at the other end (front)
2. First element to be inserted is the first one to be deleted.
3. So it’s a FIFO list
4. Enqueue = when element is inserted in queue
5. Dequeue = when element is deleted from queue
6. Underflow = trying to dequeue an empty queue
7. Overflow = trying to enqueue an element into a full queue

What is a Queue?



Applications and Implementations

1. Applications
   - To schedule OS jobs
   - Any first come first serve scenario etc
2. Implementations:
   - Array based
   - Linked list based

Queue: Array Based Implementation

1. Structure:

```
struct queue
{
    int arr[ MAX ];
    int front, rear;
};
```

1. Possible functions:

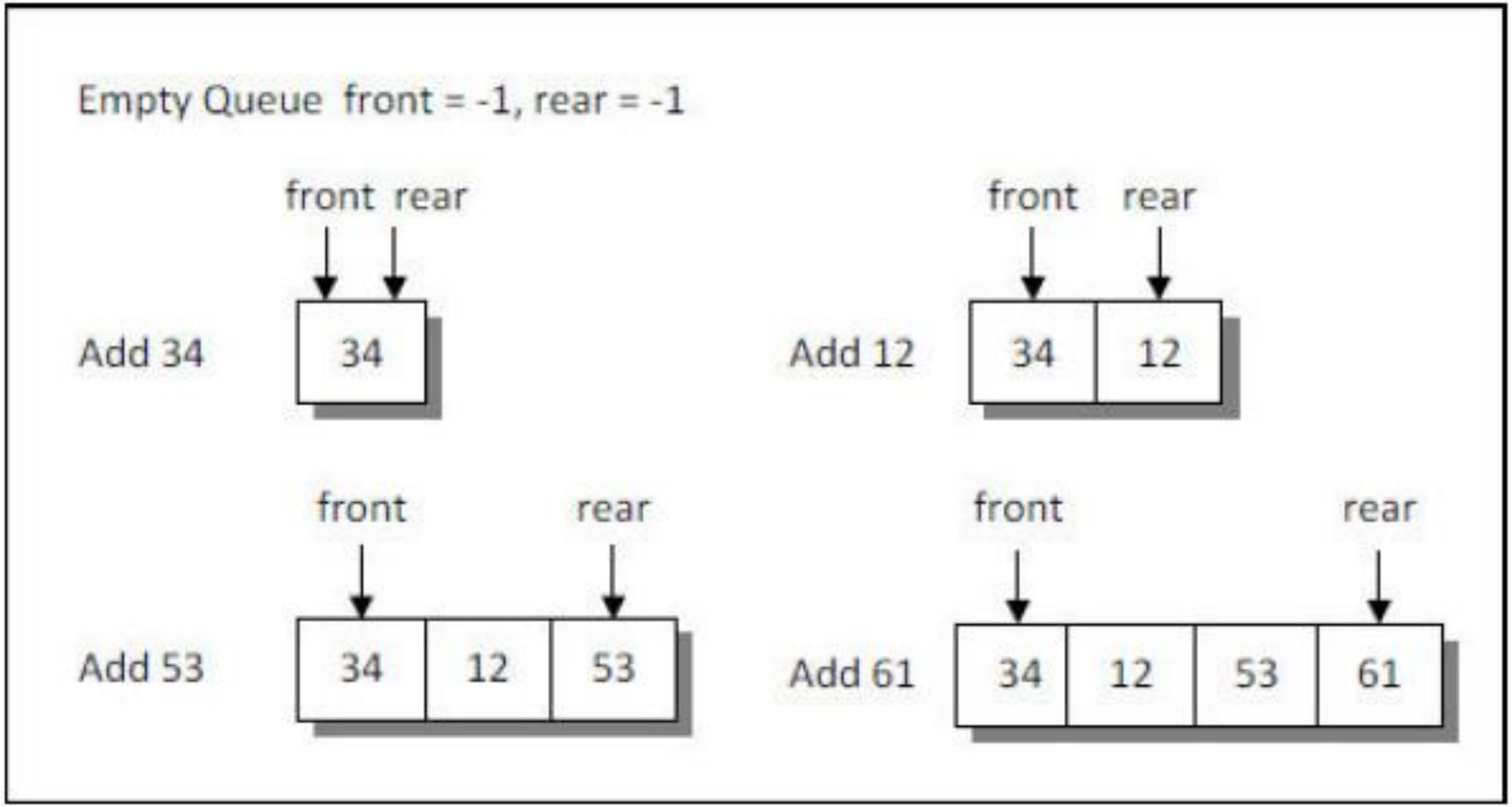
- void initq (struct queue *);
- void addq (struct queue *, int);
- int delq (struct queue *);

Queue: Array Based Implementation

1. /* intializes the queue */

```
void initq (struct queue *pq)
{
    pq -> front = -1;
    pq -> rear = -1;
}
```

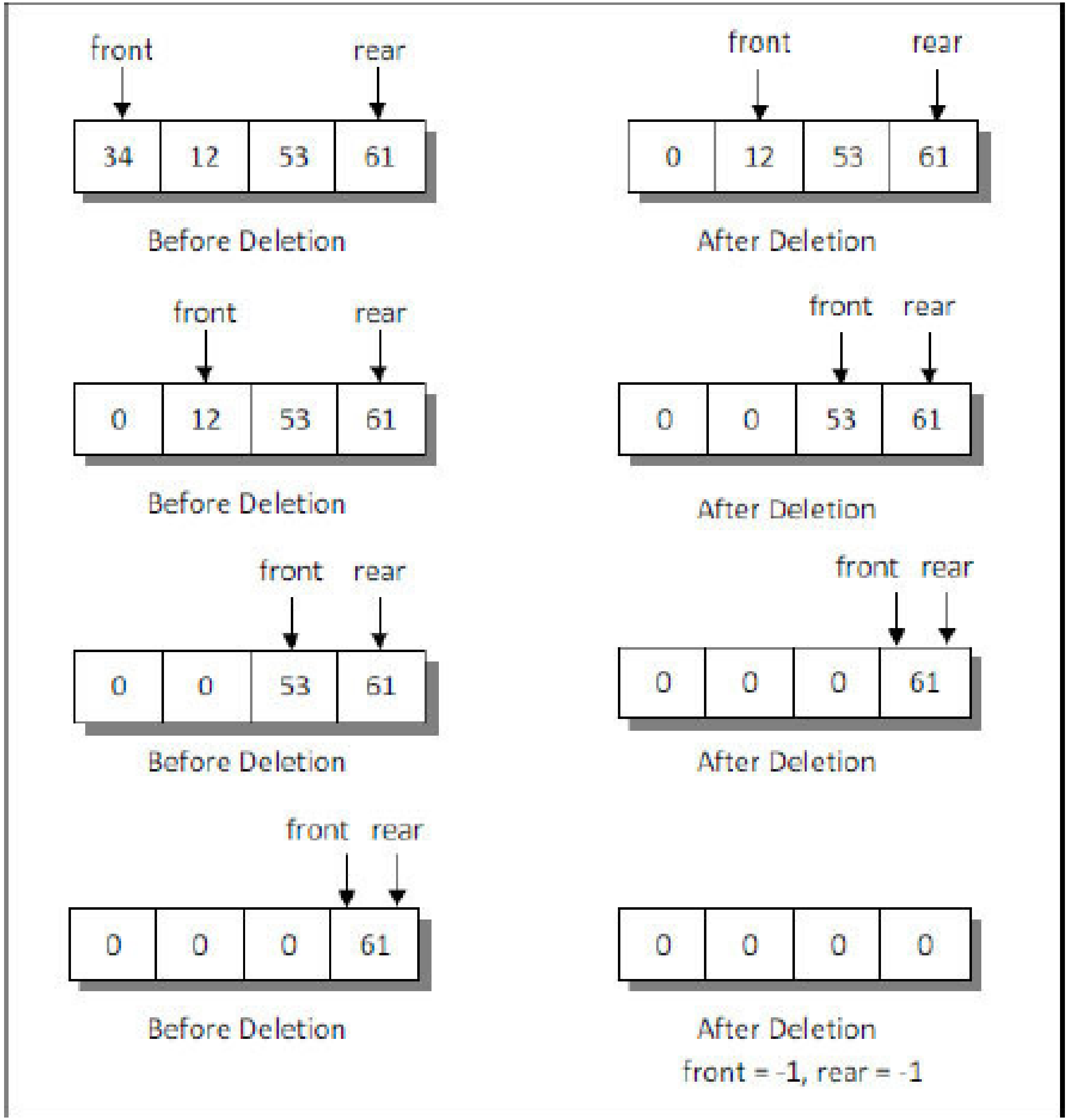
Enqueue Procedure



EnQueue: Array Based Implementation

```
1.  /* adds an element to the queue */
void addq (struct queue *pq, int item)
{
    if (pq->rear == MAX - 1) \\ overflow check
    {
    printf (“Queue is full\n”);
    return;
    }
    pq->rear++;
    pq->arr[ pq->rear ] = item;
    if (pq->front == -1)
        pq->front = 0;
}
```

Dequeue Procedure



DeQueue: Array Based Implementation

```
int delq (struct queue *pq){/* removes an element from the queue */
int data;
    if (pq->front == -1) {  \\ underflow check
    printf (“Queue is Empty\n”); }
    data = pq->arr[ pq->front ];
    pq->arr[ pq->front ] = 0;
    if (pq->front == pq->rear) //if there was only 1 element in the queue
        pq->front = pq->rear = -1;
    else
        pq->front++;
    return data;
}
```

DeQueue: Limitation of Array Implementation

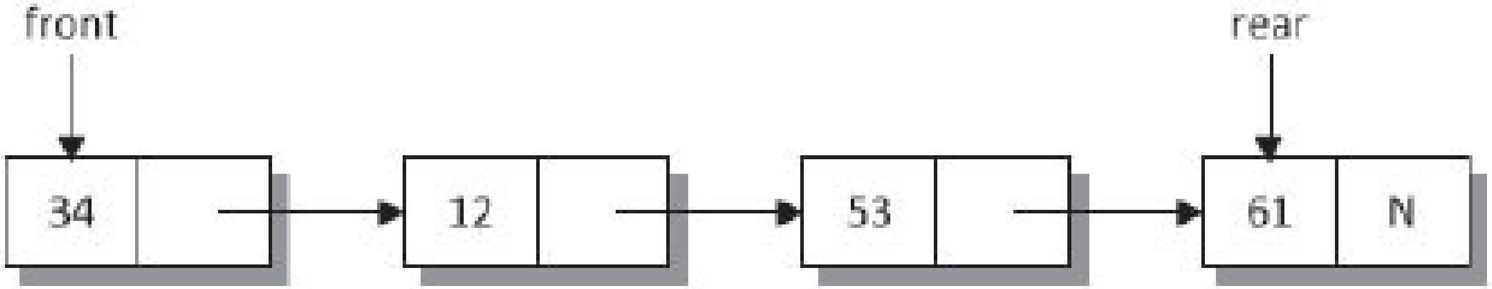
1. Suppose we go on adding elements to the queue till the entire array gets filled.
2. Now the value of rear = MAX - 1.
3. Now if we delete 5 elements from the queue, the value of front = 5.
4. If now we attempt to add a new element to the queue then it would be reported as full even though in reality the first five slots of the queue are empty.
5. To overcome this situation we can implement a queue as a circular queue, discussed later.

Recap

- Queues,
- Operations on Queue: enqueue, dequeue
- Full and Empty queues
- Overflow and underflow
- Array implementation of queues

Queue: Linked List Implementation

1. Space for the elements in a linked list is allocated dynamically, hence it can grow as long as there is enough memory available for dynamic allocation.



2. Enqueue operation implemented by inserting element at the end of LL
3. Dequeue operation implemented by deleting an element at the beginning of LL

Queue: Linked List Implementation

```
struct node
{
    int data;
    struct node *link;
};
```

```
struct queue
{
    struct node *front;
    struct node *rear;
};
```

Functions:

1. void initqueue (struct queue *);
2. void addq (struct queue *, int);
3. int delq (struct queue *);

Queue: Linked List Implementation

```
1.  /* initializes the front and rear pointers*/
void initqueue (struct queue *q)
{
    q -> front = NULL;
    q -> rear = NULL;
}
```

Queue: Linked List Implementation

```
void addq (struct queue *q, int item){ //adds an element to the queue
    struct node *temp;
    temp = (struct node *) malloc (sizeof (struct node));
    if (temp == NULL) printf (“Queue is full\n”);
    temp -> data = item;
    temp -> link = NULL;
    if (q -> front == NULL){
        q -> rear = q -> front = temp; return;}
    q -> rear -> link = temp;
    q -> rear = q -> rear -> link;
}
```

Queue: Linked List Implementation

```
int delq (struct queue * q){/* removes an element from the queue */
    struct node *temp;
    int item;
    if (q -> front == NULL){
        printf (“Queue is empty\n”);
        return NULL;}
    item = q -> front -> data;
    temp = q -> front;
    q -> front = q -> front -> link;
    free (temp);
    return item;
}
```

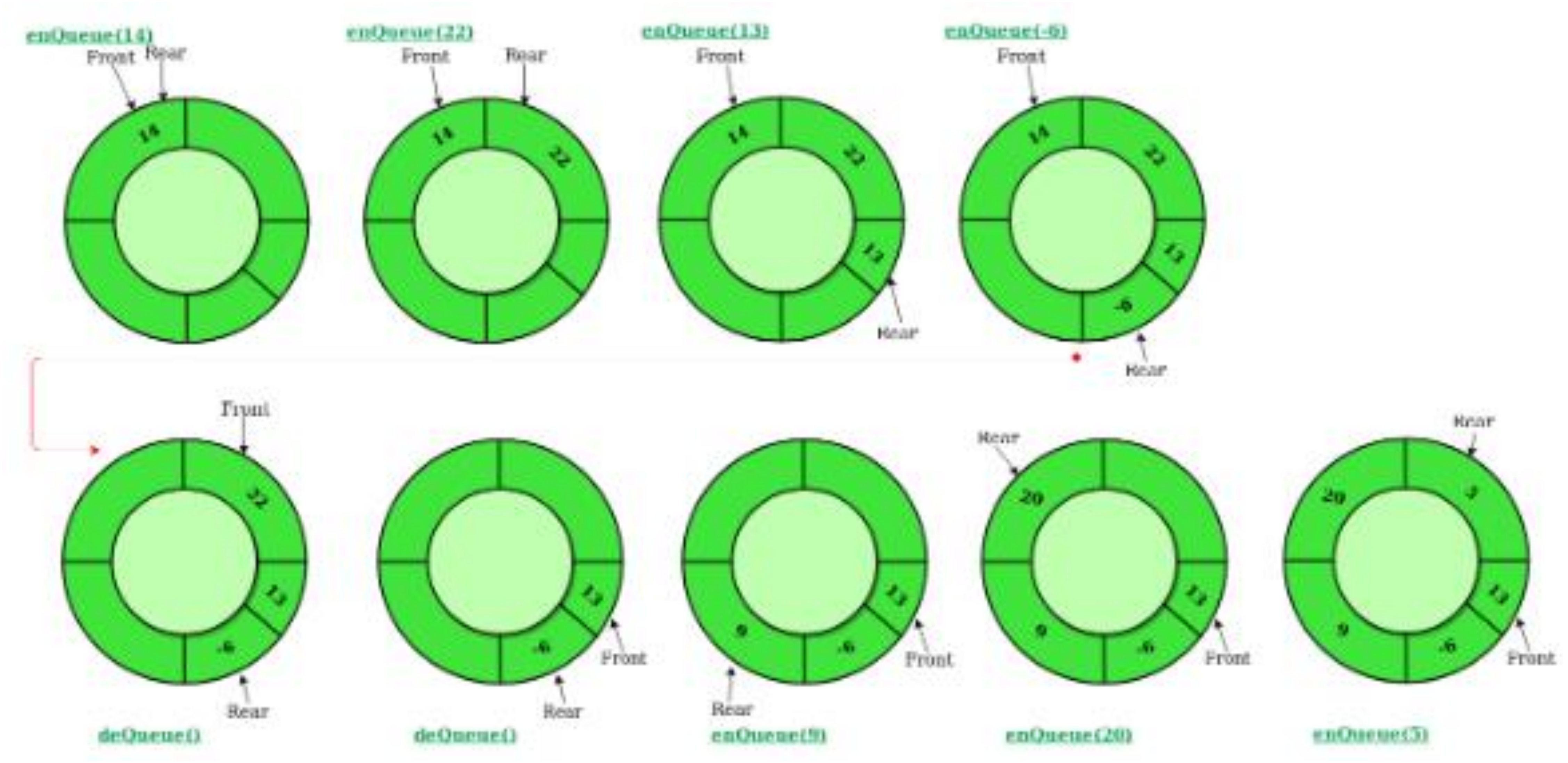
Circular Queue

1. The queue that we implemented using an array suffers from limitation that the queue is reported as full (since rear has reached the end of the array), even though in actuality there might be empty slots at the beginning of the queue.
2. To overcome this limitation we can implement the queue as a circular queue.
3. Here as we go on adding elements to the queue and reach the end of the array, the next element is stored in the first slot of the array (provided it is free).
4. The queue would be reported as full only when all the slots in the array stand occupied.

Circular Queue

1. A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle.
2. The operations are performed based on FIFO (First In First Out) principle.
3. It is also called ‘Ring Buffer’.

Circular Queue



Circular Queue – Enqueue Algorithm

To enqueue an element x into the queue of size n, do the following:

1. Check for full queue –
    - Either rear == n - 1 && front == 0
    - Or rear + 1 == front
2. Check the value of rear
    - If rear is equal to n-1, set rear to 0.
    - Else Increment rear by 1.
3. If front is -1, set front to 0.
4. Set queue[rear] to x.

Circular Queue – Dequeue Algorithm

To dequeue an element from the queue, do the following:

1. Check if the queue is empty by checking if front is -1.
2. Store the value at queue[front] in x.
3. Set queue[front] to 0
4. If front is equal to rear, set front and rear to -1.
5. Otherwise, check value of front.
    1. If front = n-1, set front to 0
    2. Else increment front by 1
6. Return x.

Circular Queue – Array Implementation

```
struct queue
{
int arr[ MAX ];
int front, rear;
};
```

Required functions

1. void initq (struct queue *);
2. void addq (struct queue *, int);
3. int delq (struct queue *);

Circular Queue

```
/* initializes an empty queue */
void initq (struct queue *pq)
{
    int i;
    pq->front = pq->rear = -1;
    for (i = 0; i < MAX; i++)
        pq->arr[ i ] = 0;
}
```

Circular Queue

```
void addq (struct queue *pq, int item){ //adds an element to the queue
if ((pq->rear == MAX - 1 && pq->front == 0) || (pq->rear + 1 == pq->front)){
    printf (“Queue is full”); return;}
if (pq->rear == MAX - 1)
    pq->rear = 0;
else
    (pq->rear)++;
pq->arr[ pq->rear ] = item;
if (pq->front == -1)
    pq->front = 0;
}
```

Circular Queue Deletion - Pseudocode

```
int delq (struct queue *pq){/* removes an element from the queue */
int data;
if (pq->front == -1){printf (“Queue is empty\n”);return NULL;}
data = pq->arr[ pq->front ];
pq->arr[ pq->front ] = 0;
if (pq->front == pq->rear){pq->front = -1;pq->rear = -1;}
else{
    if (pq->front == MAX - 1)
    pq->front = 0;
    else
    (pq->front)++;}
return data;
}
```

From here

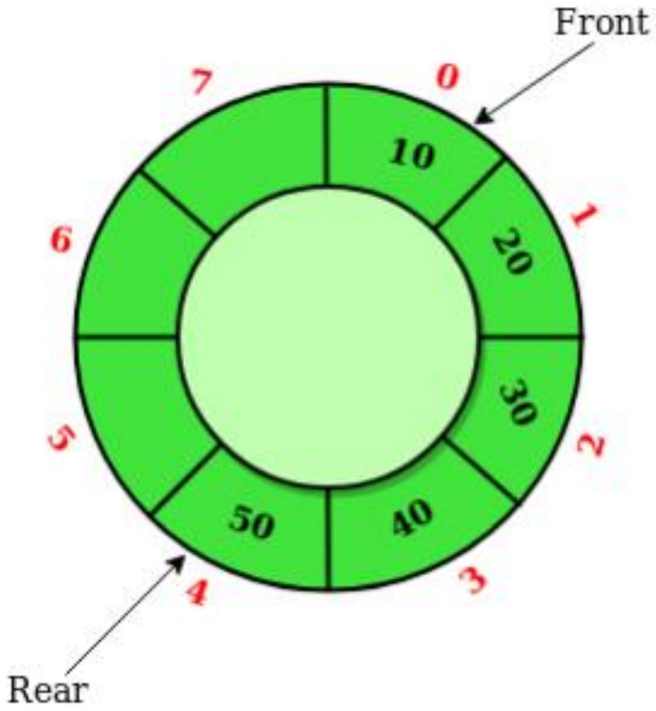
Recap

- Linked List implementation of queues
- Circular queues - insertion

Circular Queue – Enqueue Algorithm

To enqueue an element x into the queue of size n, do the following:

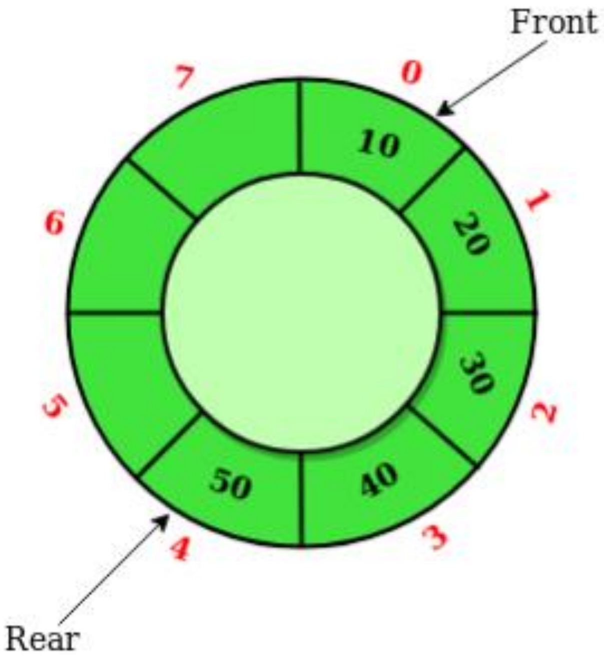
1. Check for full queue –
   - Either rear == n - 1 && front == 0
   - Or rear + 1 == front
2. Check the value of rear
   - If rear is equal to n-1, set rear to 0.
   - Else Increment rear by 1.
3. If front is -1, set front to 0.
4. Set queue[rear] to x.



Circular Queue – Dequeue Algorithm

To dequeue an element from the queue:

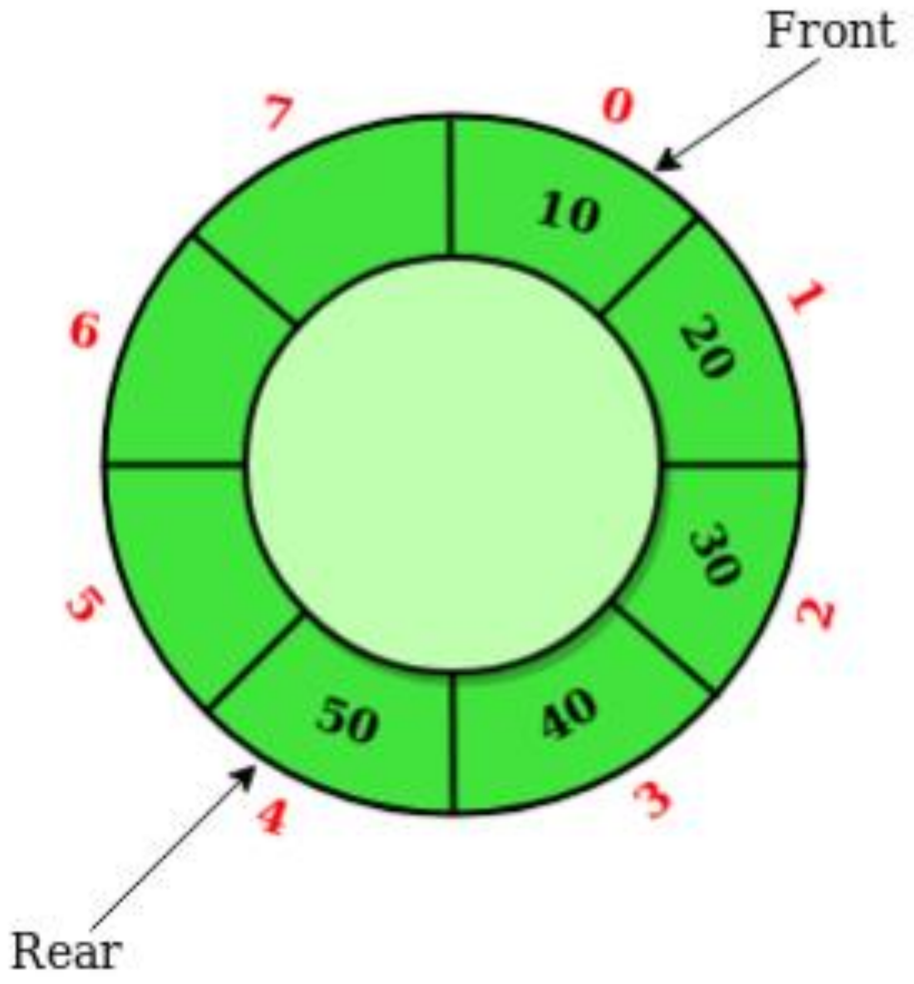
1. Check if Q is empty: if (front == -1)
2. If not: X = Q[front] // to be deleted
3. Set Q[front] = 0
4. Check if Q has only 1 element left (deleting last element)
   - If (front == rear): Set front = rear = -1 //empty Q
5. Otherwise, check value of front.
   - If (front == n-1): set front = 0 // front pointing at Q[0] now.
   - Else set front = (front + 1)
6. Return x.



Circular Queue – Enqueue and Dequeue

If this is the initial circular queue, indicate the positions of front and rear and the circular queue values, after each of the following operations:

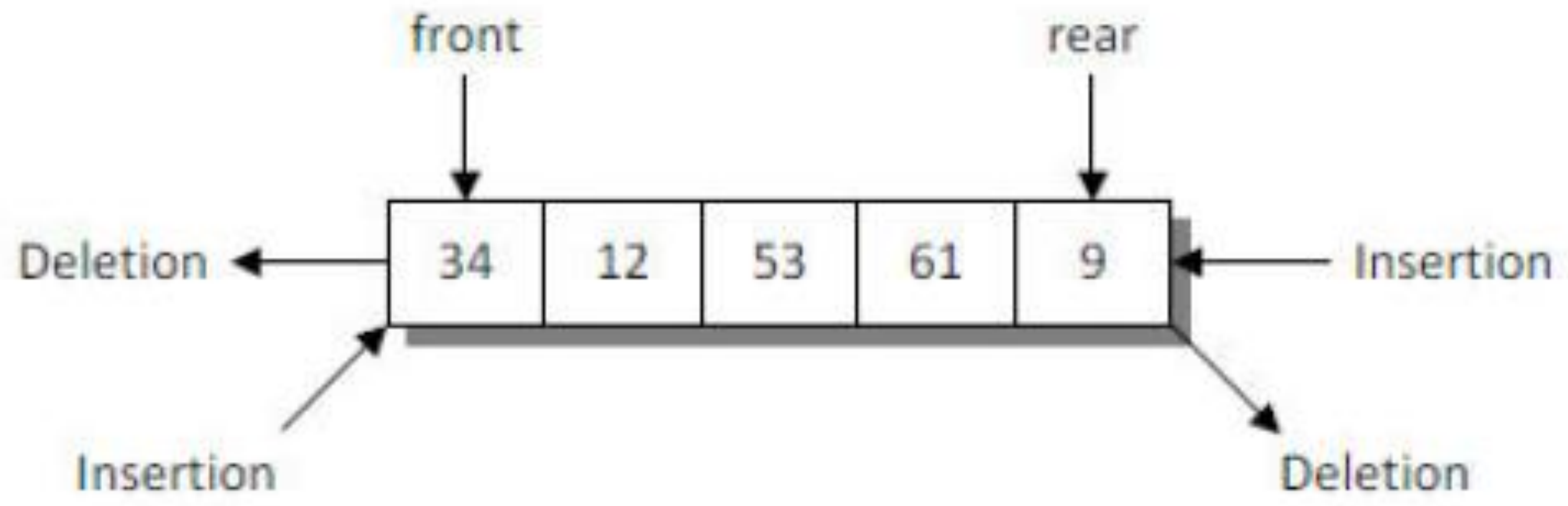
1. Enqueue 60
2. Enqueue 70
3. Enqueue 80
4. Enqueue 90
5. Dequeue
6. Dequeue
7. Enqueue 90



Double ended Queue

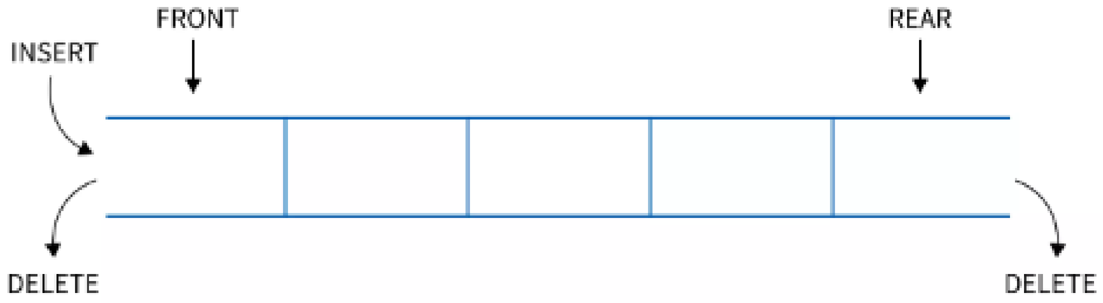
Deque

1. Deque or Double Ended Queue is a generalized version of Queue data structure that allows insert and delete at both ends

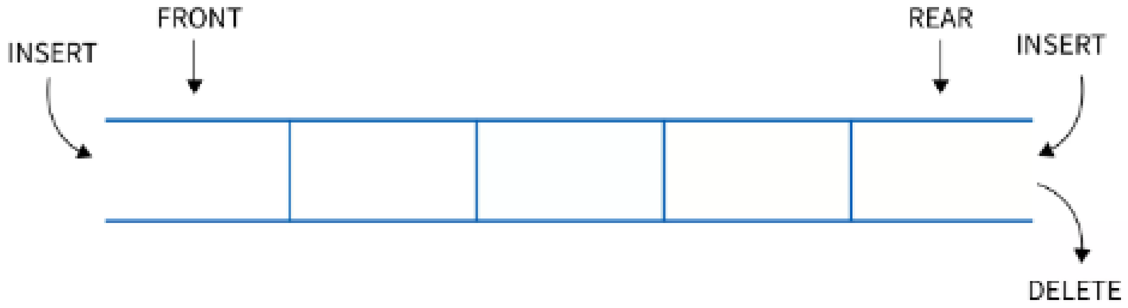


Deque

1. There are two variations of a deque—
   - an **Input-restricted deque** and
   - an **Output-restricted deque**.
2. An Input restricted deque restricts the insertion of elements at one end only, but the deletion of elements can be done at both the ends of a queue.
3. An output-restricted deque, restricts the deletion of elements at one end only, and allows insertion to be done at both the ends of a deque.



INPUT RESTRICTED DOUBLE ENDED QUEUE



OUTPUT RESTRICTED DOUBLE ENDED QUEUE

Deque

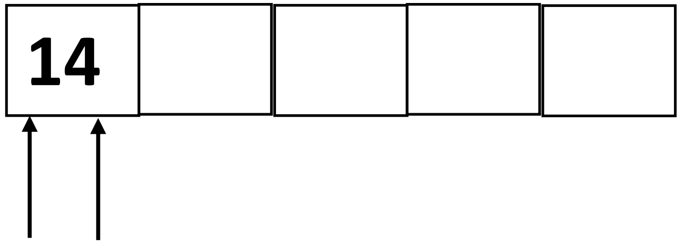
1. Four possible operations:
    - insertFront(): Adds an item at the front of Deque.
    - insertRear(): Adds an item at the rear of Deque.
    - deleteFront(): Deletes an item from front of Deque.
    - deleteRear(): Deletes an item from rear of Deque.

Array implementation of Deque:

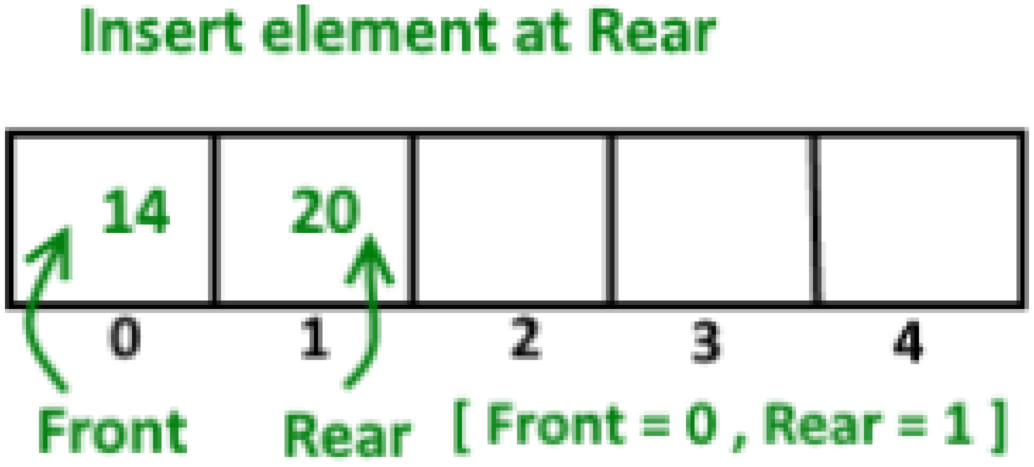
1. Create an empty array of size N
2. Initialize front = -1 , rear = -1
3. For the very first insertion, consider it to be inserted at the rear. Update rear accordingly. Update front to 0.
   - So after the first insertion, front and rear both are set to 0.
4. For each operation, we update front and rear accordingly.
5. Generally,
   - Insertion at rear happens from left to right of the array
   - Insertion at front happens from right to left of the array

Deque - Insert element at rear

1. First check deque if is Full or Not. Proceed if not full. //what is this condition?
2. IF (rear == N-1) : then set rear = 0 ; // **insertion at rear happens from left to right**
3. Else: rear = rear + 1
4. Insert current data into this rear index: Arr[rear] = data
5. Front remain same.



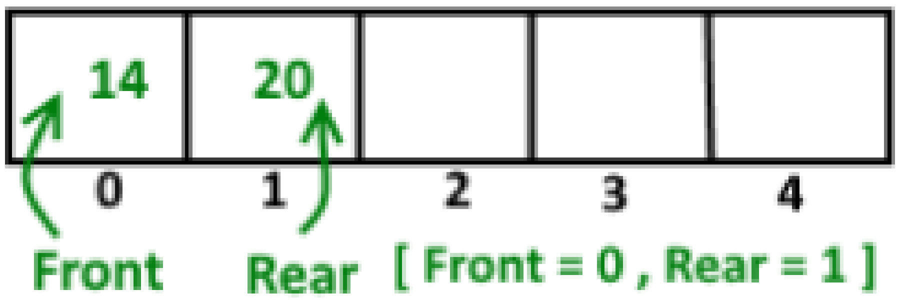
front, rear



Deque: Insert Elements at the front

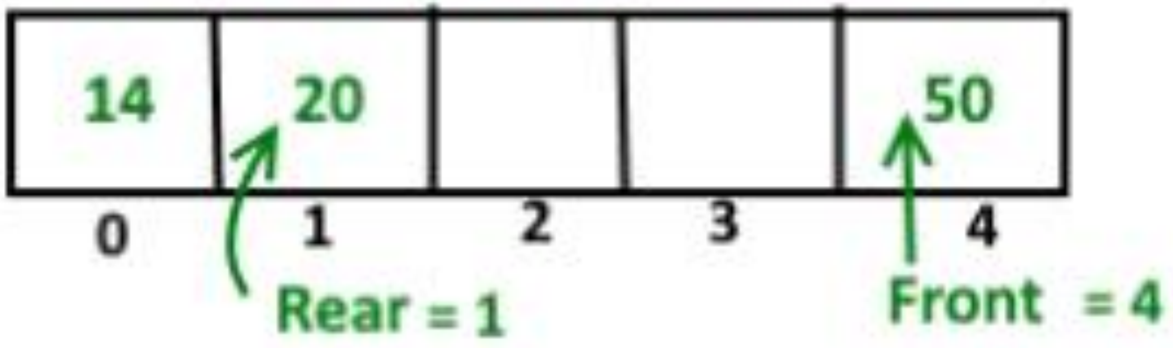
1. First check deque if Full or Not. Proceed if not full.
2. IF front == 0: then set front = N – 1 // **insertion at front happens from right to left**
3. Else: front = front - 1
4. Insert current data into this front index: Arr[front] = data
5. Rear remain same

Insert element at Rear

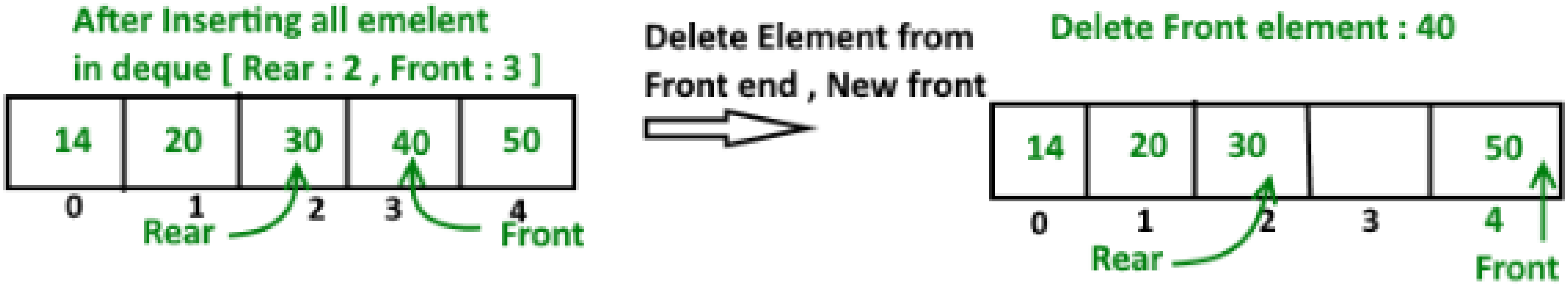


Insert element at Front end

Now Front points last index



Deque: Delete Element From rear/front



After Inserting all emelent in deque [Rear : 2 , Front : 3]

14	20	30	40	50
0	1	2	3	4

Rear

Front

Delete Element from Front end , New front

Delete Front element : 40

14	20	30		50
0	1	2	3	4

Rear

Front

Priority Queue

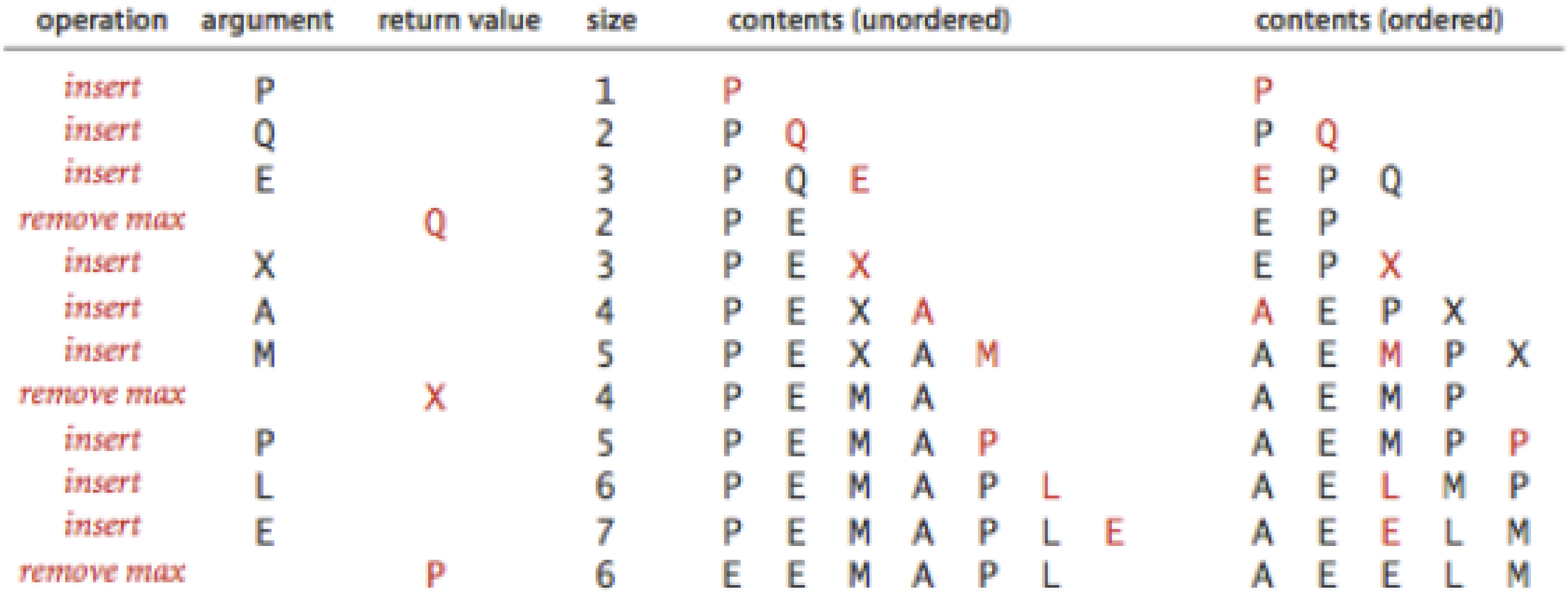
Priority Queue

1. A priority queue is a collection of elements where the **elements are stored according to their priority levels.**
2. The order in which the elements should get added or removed is decided by the priority of the element.
3. Properties:
   a) Every item has a priority associated with it.
   b) An element with high priority is dequeued before an element with low priority.
   c) If two elements have the same priority, they are served according to their order in the queue.

Priority Queue

1. How is Priority assigned to the elements in a Priority Queue?
2. Generally, the value of an element is considered for assigning the priority.
3. Types of Priority Queue:
    a) **Ascending Order Priority Queue**: Element with a lower value is given a higher priority
    b) **Descending order Priority Queue**: Element with a higher value is given a higher priority

Priority Queue



operation	argument	return value	size	contents (unordered)	contents (ordered)
insert	P		1	P	P
insert	Q		2	P Q	P Q
insert	E		3	P Q E	E P Q
remove max		Q	2	P E	E P
insert	X		3	P E X	E P X
insert	A		4	P E X A	A E P X
insert	M		5	P E X A M	A E M P X
remove max		X	4	P E M A	A E M P
insert	P		5	P E M A P	A E M P P
insert	L		6	P E M A P L	A E L M P
insert	E		7	P E M A P L E	A E E L M
remove max		P	6	E E M A P L	A E E L M

A sequence of operations on a priority queue

Queue

Common Questions:

1. Queue Implementation using a Linked List
2. Implement a stack using the queue data structure
3. Implement queue using stack data structure
4. Implement a deque using i) array and ii) doubly linked list.

Thanks!