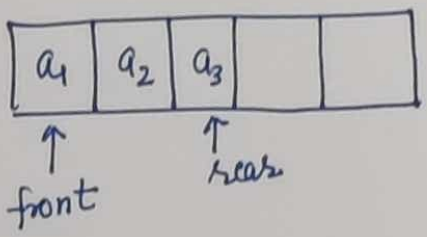
Queue

- Philosophy for insertion and deletion:

  FIFO (FCFS).

- Elements inserted at the "rear" and removed from "front".



ADT:

Operation:-

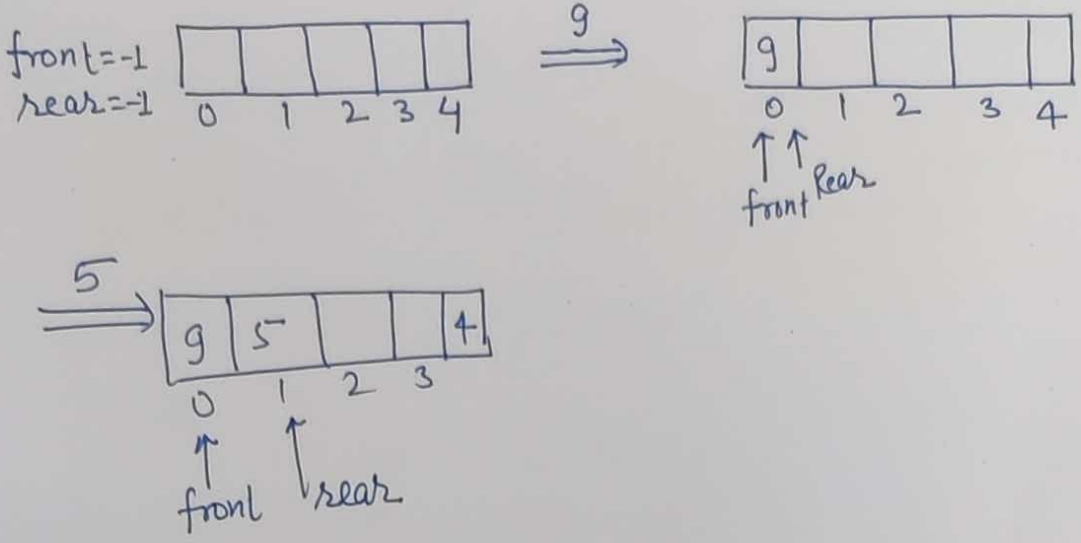
* New(): used for creation of new (empty) queue.
* Enqueue (s, x): to insert element 'x' at the rear
* Dequeue (s, x): Remove x from front
* Front() / Peak(): return the front element
* Size(): # of elements (integer)
* isFull(): Overflow (binary)
* isEmpty: Underflow (binary)

Axioms:-

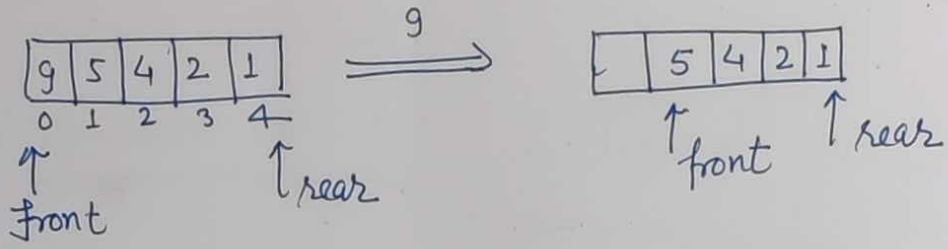
* Front (Enqueue (New(), x)) = x
* Dequeue (Enqueue (New(), x)) = New()
* Front (Enqueue (Enqueue (Q, x), y) = Front (Enqueue(Q, x))
* Dequeue (Enqueue (Enqueue (Q, x), y)) =

  Enqueue (Dequeue (Enqueue (Q, x)), y)

Example :- ENQUEUE



- DEQUEUE -



= IMPLEMENTATION =

USING STATIC ARRAY :-

- Realizes queue of maximum possible size.
- front is maintained at the smallest index and rear at the max. index.

n : size of array
Q : queue.

ENQUEVE :-

```
if (rear = n-1)
    "Print overflow"
else if (front=-1 & rear=-1)
        front ← 0
        rear ← 0
    else rear ← rear+1
Q[rear] ← item
```

Precondition for Enqueue: Q has been initialized and not full

Postcondn :- new item at rear of queue.

Dequeue

Precond^n :- Q has been initialized and is not empty

Postcond^n :- front element has been removed

1. If (front = -1 || front > rear)

   Print "underflow"

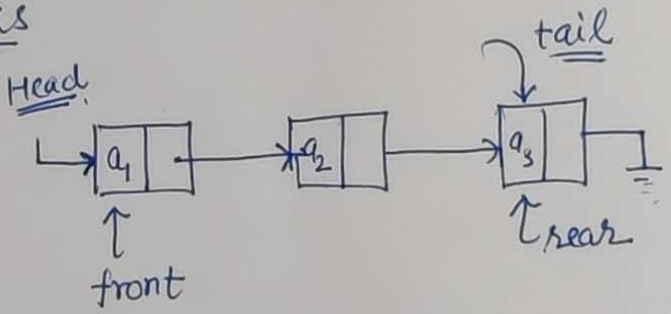
   else

   item ← Q[front]

   front ← front+1

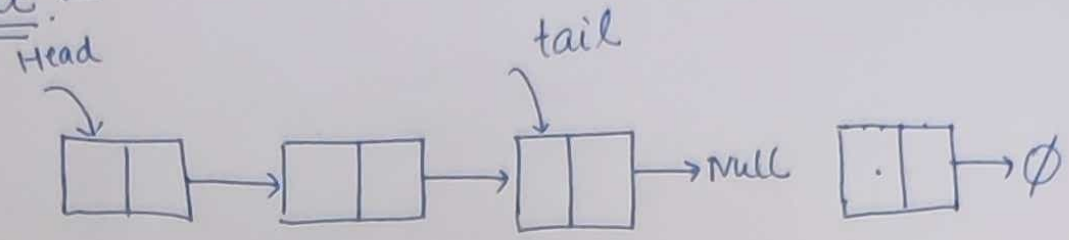
Dynamic Linked list Implementation

1. Node (data, Ptr) connected in a chain by links

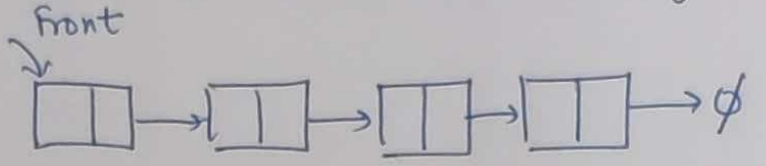


[Head of the list : front
Tail ——— : rear]

Enqueue:-



- Create a node at tail
- add it and move the tail reference.



Algo:-

1- create a node pointer, say. temp.

2. temp [data] ← item

3. temp [next] ← ϕ

4. if (front = ϕ)

front ← temp

rear ← temp

else

rear [next] ← temp

rear ← temp

Dequeue:-

If (front = Null)

Print "underflow"

else . Initialize a node pointer temp & set it to front

if (front = rear)

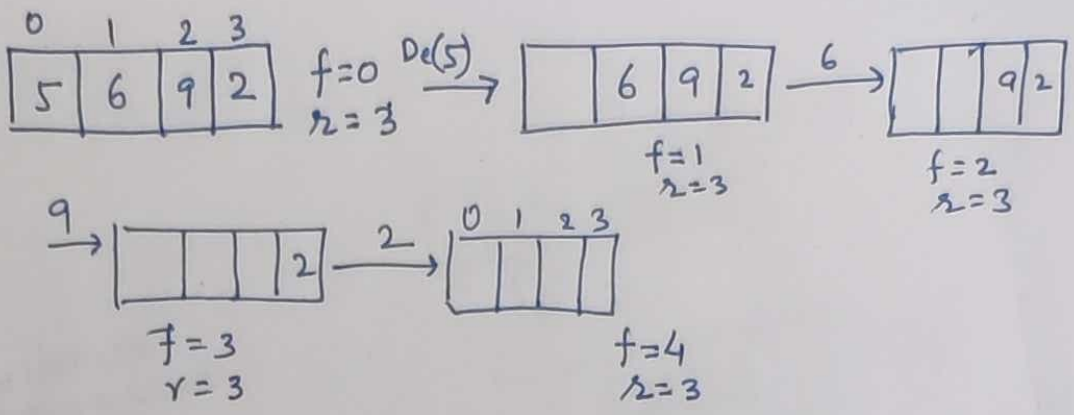
front = rear = ϕ

else

front ← front [next]

release memory location pointed by temp.

- Problem with simple Queue -



⇒ rear = n-1 : means overflow

⇒ But, queue is empty.

CIRCULAR QUEUES / Double-Ended Queue

Support Insertion & Deletion from front & rear.

→ Operation:

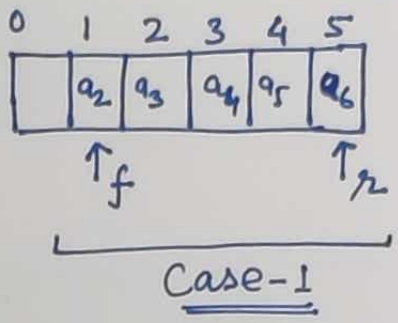
InsertFirst(S,x) : insert at begining of queue.
InsertLast(S,x) : ——— end ———
RemoveFirst(S) : remove the first
RemoveLast(S) : ——— last ———
First(S) : return first element
Last(S) : return last ———

Issue(s) In implementing Queue with "simple" array :-

ENQUEUE :-

To do enqueue :.

- function :- Add new item to rear of the queue
- Precond^n :- queue is not full
  $r < n-1$
- Postcond^n :- new item should be added to rear of queue.



Case-1

* Case 1 :- - Insertion & Deletion is O(1) time but created new problem.
  - Can't insert new element. even place(s) available at the start of array.

  * Soln :. Allow the queue to "wrap around"
    ⇓
    use "circular array"

$f = r = ??$

→ ⇓ means ⇒ Queue is empty
sometimes :- Queue is full

⇓

Ambiguity whether Q is empty or full.

(means once I have deleted all thing)

Overcome the Ambiguity :.

One way :. Q can never hold "n-1" elements., where is the size of array.

Size(Q) $(n-f+r) \bmod n$ - correct for both the cases: normal array or circular array.

⇒ When "$f = r$", Q will be empty.

ENQUEUE USING ARRAY IN CIRCULAR FASHION

Precondition: queue is not full. means.

$(n-f+r) \bmod n \neq n-1$

or

size() $\neq n-1$

Post condition: $r \leftarrow (r+1) \bmod n$

```
Enqueue(Q, item)
    if (n-f+r) mod n = n-1
        return "Overflow cond^n"
    else
        Q[r] ← item
        r ← (r+1) mod n
```

```
Dequeue(Q)
    If f = r
        return "underflow cond^n"
    else
        temp ← Q[f]
        Q[f] ← Null
        f ← (f+1) mod n
    return "temp"
```

Precond^n. $f \neq r$

Post:- $f \leftarrow (f+1) \bmod n$