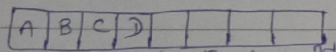
04 Queue

* Linear ds in which dlt can only occur at 1 end → front & ins occur at rear
* FIFO , eg → real-life queue.

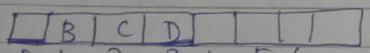
* Basic operations =
1) Enqueue → ins (rear)
2) Dequeue → dlt (front)

→ representation of Q as arrays =

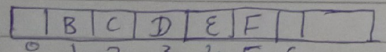
(a) Front: 0, Rear: 3



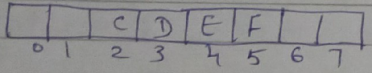
(b) F: 1, R: 3



c) F: 1, R: 5



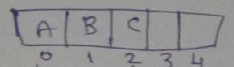
d) F: 2, R: 5



Q is empty if Front = Rear = -1.
Q is full if Front = 1 & rear = max-1.

* INSERT (QUEUE, N, FRONT, REAR, ITEM).
1) Start.
2) If REAR = N-1 then print overflow
3) Return
after if 4) Set QUEUE [REAR] = ITEM.
5) Set REAR = REAR+1
6) Return.

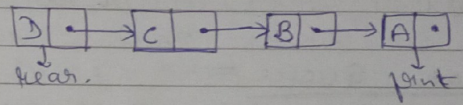
* DELETE (QUEUE, N, FRONT, REAR, ITEM)
1. Start
2. If FRONT >= N OR FRONT > REAR then
Print underflow.



3. Return
after if 4. Set ITEM = QUEUE[FRONT]
5. Set FRONT = FRONT + 1
6. Return ITEM.

→ Representation of Q as L.S =

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



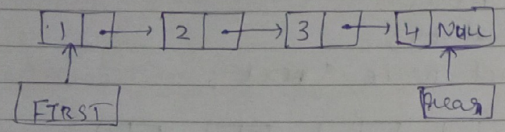
* INSERT (QUEUE, FRONT, REAR, ITEM)
1) Create NEWNODE
2) Set INFO(NEWNODE) = ITEM.
3) If FRONT = REAR = NULL then
FRONT = REAR = NEW
4) Else
Set REAR[NEXT] = NEWNODE
Set REAR = NEWNODE
5) Return.

* DELETE (QUEUE, FRONT, REAR, ITEM)
1) If FRONT = REAR = NULL
2) display underflow
3) return
after if 4) Set FRONT = NEXT(FRONT)
5) Return

* The disadv of using an array to represent

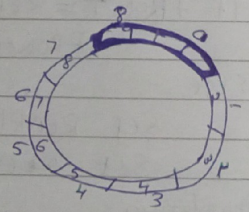
a Q is that the array must be specified to hue a fixed size.

* If the array size cannot be known, then L.S representation is used.
* HEAD pointer of L.S is used as the FRONT.
* Anter pointer called REAR to store the address of last elemnt of Q.



=> Circular Queue =

* It is a sple type of Q where last elemnt is connected back to the 1st elemnt, thus forming a circle.
* In C. Q, the 1st index comes right after the last index.



* ins in C. Q = INSERT (QUEUE, MAX, FRONT, REAR, ITEM).

1) Start
2) if FRONT = 1 and Rear = max - 1
   print overflow.
3) if Front = -1 and Rear = -1
   Set front = 1 and rear = 1.
4) else
   if rear = max - 1
   Set rear = -1
   else
   Set rear = rear + 1
6) Set QUEUE [REAR] = ITEM.
5) Return.

* dlt in C. Q =

DELETE (QUEUE, MAX, FRONT, REAR, ITEM).

1) if Front = NULL then
   print underflow.
2) Return
3) Set ITEM = QUEUE [FRONT]
4) If FRONT = REAR then
5) Set front = NULL
   Set REAR = NULL
6) else
   if front = max - 1 then
   set front = 1
   else
   set front = front + 1
7) Return.

* C. Q representation =

a) empty array — F: -1, R: -1 — [ ] [ ] [ ] [ ] (0 1 2 3)

b) A, B, C added — F: 0, R: 2 — [A] [B] [C] [ ] (0 1 2 3)

c) A deleted — F: 1, R: 2 — [ ] [B] [C] [ ] (0 1 2 3)

d) d inserted — F: 1, R: 3 — [ ] [B] [C] [d] (0 1 2 3)

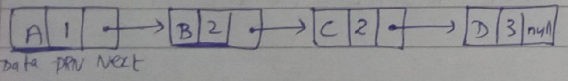
e) B, dlt
   C dlt
   d dlt — F: -1, R: -1 — [ ] [ ] [ ] [ ] (0 1 2 3)

Q becomes empty.

⇒ **priority queue** =

* It is a collection of elemnts, in which each elemnt has been assigned a priority value.
* The order in which the elemnts will be processed is decided by the priority of the elemnt.
* An elemnt with higher priority is processed 1st.
* If elemnts with same priority occur, they are processed according to the order in which they were added to the Q.

* **linked representation of p.Q.** = when a p.q is implemented with a LS each node will contain 3 parts –
  a) Data
  b) priority num of the elemnt (PRN)
  c) address of nxt elemnt.

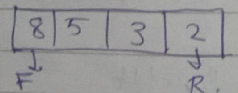


* **2 types of p.q →**

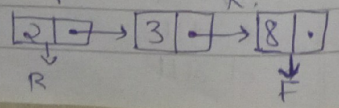
a) **max p.q.** = elemnts are ins according to their priority order & always max value is removed 1st frm the Q.

b) **min p.q** = It is similar to max p.q except for removing min elemnt 1st instead of max elemnt.

* Array representation =>

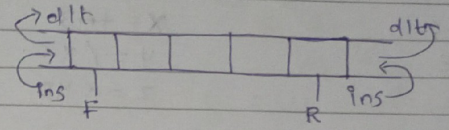


* L.S representation =>



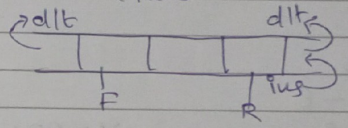
⇒ **Deque** → (Double-Ended Queue).

* ordered collection of items similar to Q.
* Here, ins & dlt can takes place at both the Front and Rear end of Q.

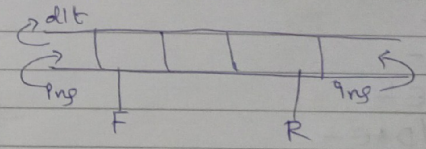


* **Represented 2 ways :**

a) **Input restricted Deque** = here ins is performed at only 1 end & dlt is performd at both ends.



b) **Output restricted Deque** = here dlt is performed at only 1 end & ins is performed at both ends.



⇒ **Application of Q** =

* CPU Scheduling
* Resource Sharing
* Disk Scheduling.

⇒ **Application of Stack** →

( )
^
* /
+ −

( )
* / (L−R)
+ (L−R)

*) Evaluating exp →

1) infix to prefix = (Same precedent വന്നാൽ push ചെയ്യുക)

K+L−M*N+(O^P)*W/U/V*T+Q.

A) Reverse it.

Q+T*V/U/W*)P^O(+N*M−L+K.

Scan it.

Stack: (^) — bracket-ൽ ഉള്ളിൽ ഉള്ളത് pop ചെയ്യുക.

operator → ① stack: * / / * +

② stack: * / / * +

+ → +<* so, pop until same precedence

exp (operand).
QTVUWPO^* //*NM *LK+−++.

③ stack: * + +

− → −<*

stack: + − + +   Then pop all.

⇒ Then reverse,

⇒ ++−+KL*MN*//*^OPWUVTQ

② infix to postfix = (Same precedence വന്നാൽ pop ചെയ്യുക.)

A+B*C/(D−E+A^E.

A) stack: * +   / = * pop.   stack: / +

exp: ABC*

③ infix to postfix =

A*(B+C)−(D+G/K)

A)

Symbol	Stack	Postfix exp.
	empty.	
A		A.
*	*	A.
(	*(	A
B	*(	AB
+	*(+	AB
C	*(+	ABC.
)	*	ABC+
−	− (− lower.p)	ABC+*
(	−(	ABC+*
D	−(	ABC+*D
+	−(+	"
G	−(+	ABC+*DG
/	−(+/	
K	−(+/	ABC+*DGK.
)	.	ABC+*DGK/+−

ABC+*DGK/+− //

(3) Evaluating prefix exp =

exp → + - * 2 2 / 16 8 5

reverse → 5 8 16 / 2 2 * - +

Symbol	Stack
5	5
8	5 8
16	5 8 16
/	5 2. (16/8)
2	5 .2 2
2	5 2 22
*	5 2 4
-	5 2 (4-)
+	7 (5+2)

=> (7)

4) Evaluating postfix exp =

exp = 5 6 2 + * 12 4 / -

Symbol	Stack
5	5
6	5 6
2	5 6 2
+	5 8
*	40. (5*8)
12	40 12
4	40 12 4
/	40 3 (12/4)
-	37 (40-3)

=> 37 //