C function →

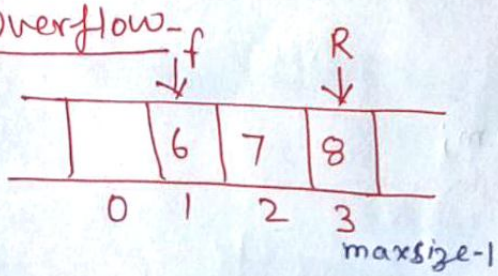
```
#define maxsize 30
int DQ[maxsize], front=-1, rear=-1;
void insert_at_front()
{
  int item;
 if (front == 0)
 {
    printf("Overflow");
    exit(0);
 }
 if (front==-1)
  front= rear= 0;
 else
    front= front-1;

 printf("Enter item");
 scanf("%d", &item);
  DQ[front]= item;

}
```

2) Insertion at rear end → concept →

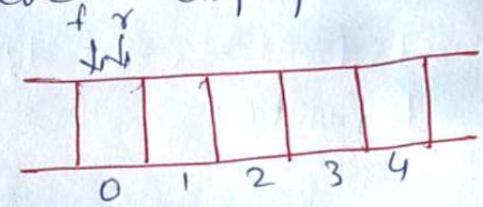
Case 1 → Overflow




```
if (rear == maxsize-1)
  printf("Overflow");
```

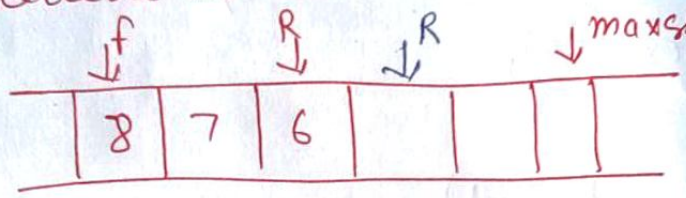
case2→ Queue is empty

f=-1
r=-1




```
if (rear == -1)
    front = rear = 0;
```

case3; Queue is neither empty nor full at rear end.




```
rear = rear + 1
```

Algorithm→

```
Insert_at_rear( )
```

1. read item
2. if (~~front~~ rear == maxsize-1) then
   write "Overflow" & exit
3. if (rear == -1) then
   front = rear = 0;
   else
   rear = rear + 1;
   [End of if]
4. DQ[rear] = item;
5. Exit.

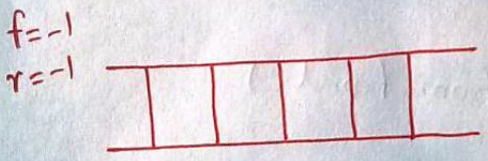
Q function→

```
void insert_at_rear()
{
  if (rear == maxsize-1)
  {
    printf("overflow");
    exit(0);
  }
  if (rear == -1)
   front = rear = 0;
  else
    rear = rear+1;
  DQ[rear] = item;
}
```

3) Deletion from front-

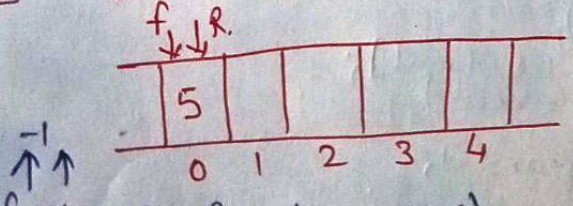
concept→

Case 1→ Underflow



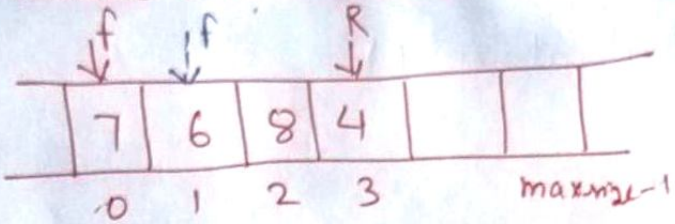

```
if (rear or front == -1)
   printf("Underflow");
```

Case 2→ Queue contains only one item




```
if (front == rear)
   front = rear = -1
```

Case 3 - Queue contains more than one item



front = front + 1

Algorithm →

Deletion_from_front()

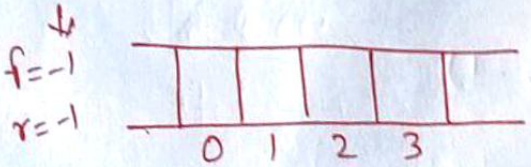
1. if (front == -1) then
   write "Underflow" & exit
2. item = DQ[front]
3. if (front == rear) then
   front = rear = -1
   else
   front = front + 1
4. write item, "is deleted"
5. Exit

C function →

```
void deletion_from_front()
{
  int item;
  if (front == -1)
  {
    printf("Underflow");
    exit(0);
  }
  item = DQ[front];
  if (front == rear)
    front = rear = -1;
  else
    front = front + 1;
  printf("%d is deleted", item);
}
```

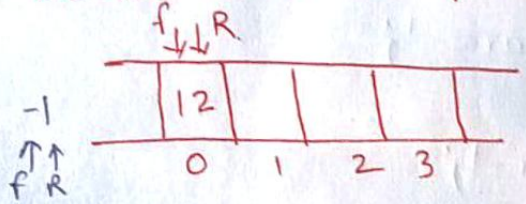
4) Deletion from rear → concept

case 1 → Underflow



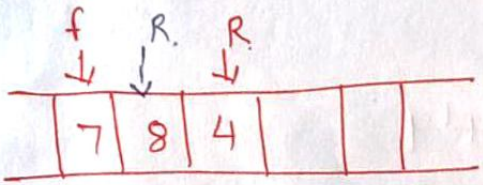

```
if (rear==-1)
   printf("underflow");
```

case 2 → Queue contains only one item




```
if(front==rear)
   front=rear=-1
```

case 3: Queue contains ~~and~~ more than one item




```
rear=rear-1
```

Algorithm → Deletion_from_rear()

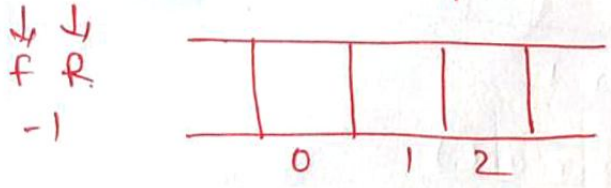
1. if (rear==-1) then
   write "Underflow" & exit.
2. item= DQ[rear]
3. if (front==rear) then
   front=rear=-1
4. else
   rear=rear-1
   [End of if]
4. write item "is deleted"
5. Exit.

c function →

```
void deletion_from_rear()
{
  int item;
  if (rear == -1)
  {
    printf("Underflow");
    exit(0);
  }
  item = DQ[rear];
  if( front == rear)
    front= rear = -1;
  else
    rear= rear -1;
  printf("%d is deleted", item);
}
```

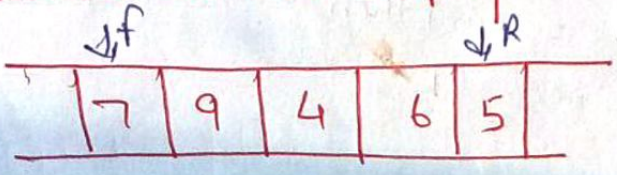
5) Traverse → concept

case 1: Queue is empty




```
if ( front == -1)
  printf("Queue is empty");
```

case 2: Queue is not empty




```
for (i=front; i<=rear; i++)
  printf("%d", DQ[i]);
```

Algorithm → traverse()

1. if (front == -1) then
   write "Queue is empty"
2. set i = front
3. repeat step 4 & 5 while (i <= rear)
4. write DQ[i]
5. set i = i+1

   [End of loop]
6. Exit

C function →

```
void traverse()
{
  int i;
 if (front == -1)
 {
   printf("Queue is empty");
    exit(0);
 }
 for(i=front; i<=rear; i++)
    printf("%d", DQ[i]);
}
```

D Queue (Linked List)

```
struct node
{
   int info;
struct node *prev;
struct node *next;
} *front = NULL, *rear == NULL;
```

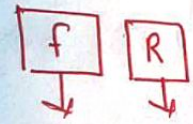
1) Insertion at front →

Concept

Case 1: Overflow

```
new = malloc()
if (new == NULL)
  printf("overflow");
```

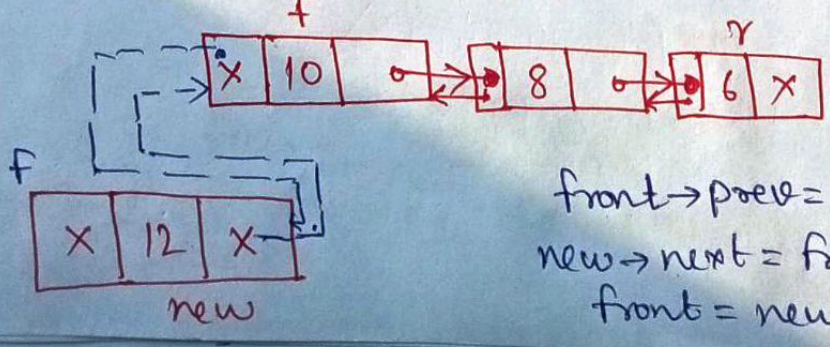
Case 2: Queue is empty



~~if(f==NULL)~~

```
if(front == NULL)
   front = rear = new
```

Case 3: Queue is not empty




```
front → prev = new
new → next = front
front = new
```

Algorithm →

(1) Insert_at_front()

1. new = AVAIL
2. if (new == NULL) then
   write "overflow" & exit
3. INFO[new] = item
4. LINKN[new] = NULL
5. LINKP[new] = NULL
6. if (front == NULL) then
   front = rear = new
   else
   LINKP[front] = new
   LINKN[new] = front
   front = new
7. exit

C function →

```
Void insert_at_front()
{
  struct node *new;
  new=(struct node *) malloc(sizeof(struct node));
  printf("Enter node info");
  scanf("%d", &new->info);
   new->prev = NULL;
   new->next = NULL;
  if (front == NULL)
    front = rear = new;
  else
  { front->prev = new
    new->next = front
    front = new
  }
}
```

Insertion at ~~end~~ rear -

Concept →

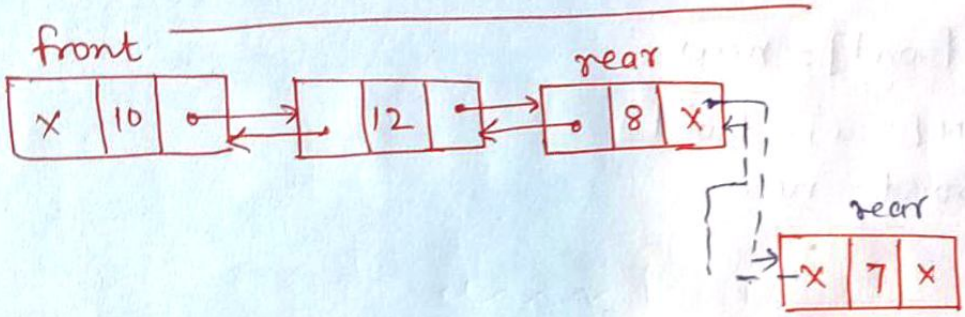
Case 1: Overflow

```
new = malloc()
if (new == NULL)
  printf("Overflow");
```

case 2: ~~Cas~~ Queue is empty

```
if (rear == NULL)
  front = rear = new
```

case 3: Queue is not empty




```
rear → next = new
new → prev = rear
rear = new
```

Algorithm →

Insert_at_rear()

1. new = AVAIL
2. if (new == NULL) then
   write "Overflow" & exit.
3. if (rear == NULL) then.
   front = rear = new
   else
   LINKN [rear] = new
   LINKP [new] = rear
   rear = new
4.

Algorithm →

Insert_at_beg()

1. new = AVAIL
2. if (new == NULL) then
   write "Overflow" & exit
3. INFO[new] = item
4. LINKN[new] = NULL
5. LINKP[new] = NULL
6. if (rear == NULL) then
   front = rear = new
   else
   rear → next = new
   new → prev = rear
   rear = new
7. Exit

C function-

```
void insert_at_beg()
{
  struct node *new;
new=(struct node *) malloc(sizeof(struct node));
if (new == NULL)
{ printf("Overflow");
  exit(0);
}
printf("Enter node info");
scanf("%d", &new->info);
new->prev = NULL;
new->next = NULL;
if (rear == NULL)
  front = rear = new
else
```

```
{
    rear→next = new;
    new→prev = rear;
    rear = new;
}
}
```

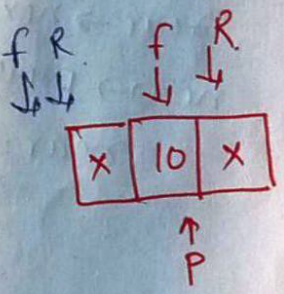
3) Deletion from front-

concept→

case 1: Underflow

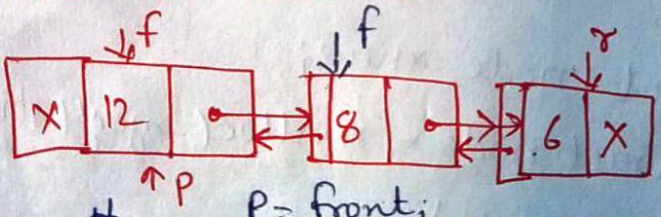
```
if (front == NULL)
    printf("Underflow");
```

case 2: Only one node




```
P = front
if (front==rear)
    front = rear = NULL;
    deletecp; freecp)
```

case 3: more than one node




```
P = front;
front = front→next;
front→prev = NULL;
deletecp); freecp);
```