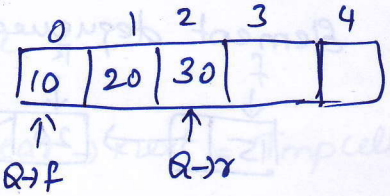
```
else
    Q->front++;
}
}
```

(i)



Dequeue (1000);

$x = Q \rightarrow Array[Q \rightarrow front]$

$= Q \rightarrow Array[0]$

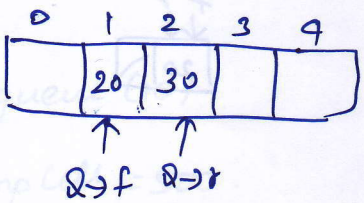
(ie) $x = 10$.

$Q \rightarrow Size--$ (ie) $Q \rightarrow Size = 3 - 1 = 2$.

if ($Q \rightarrow front == Q \rightarrow rear$)

$(0 == 2) \Rightarrow$ false

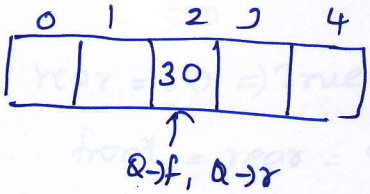
else $Q \rightarrow front++$ (ie) $Q \rightarrow front = 1$



(ii) Dequeue (1000)

$x = 20$.

$Q \rightarrow Size = 1$



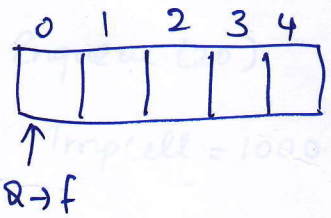
if $(1 == 2) \Rightarrow$ false

else $Q \rightarrow front++$ (ie) $Q \rightarrow front = 2$.

(iii) Dequeue (1000)

$x = 30$

$Q \rightarrow Size = 0$



if $(2 == 2) \Rightarrow$ true

$Q \rightarrow rear = -1$

$Q \rightarrow front = 0$

$Q \rightarrow Size = 0$.

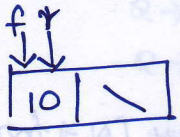
ii) Linked List Implementation of Queue ADT:

* Enqueue operation is performed at the end of the list and dequeue operation is performed at the front of the list.

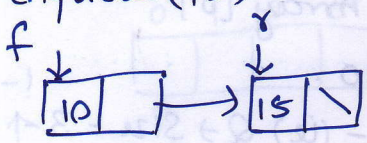
* Two pointers are used called front and rear. Front points to the node in which the element to be dequeued is present and rear points to the node in which recently inserted element is present.

Example - Enqueue

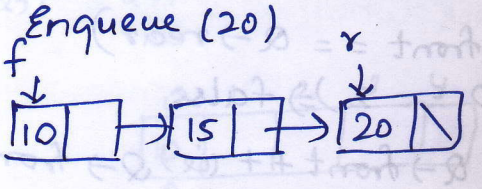
Enqueue (10)



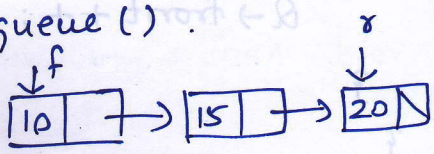
Enqueue (15)



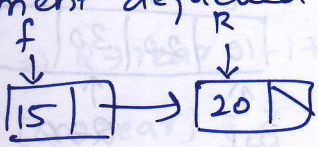
Enqueue (20)



Dequeue ()



Element dequeued is 10



Dequeue ()

Element dequeued is 15

(i) Type Declarations :

```
struct node
{
  ElementType Element;
  struct node *next;
};
struct node * front, * rear;
front = rear = NULL;
void Enqueue (ElementType X);
void dequeue();
```

(ii) Enqueue - It is used to insert an element at the end of the queue.

```
void Enqueue (ElementType X)
{
  struct node *Tmpcell;
  Tmpcell = malloc (sizeof (struct node));
  Tmpcell -> Element = X;
  Tmpcell -> next = NULL;
```

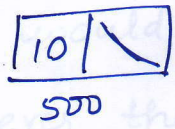
```
if (rear == NULL)
{
  front = TmpCell;
  rear = TmpCell;
}
else
{
  rear -> next = TmpCell;
  rear = TmpCell;
}
```

} for 1st enqueue

} for remaining enqueues.

eg (i) Enqueue (10).

TmpCell = 500.

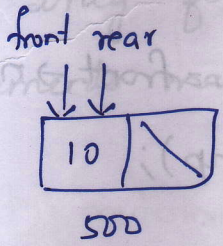


rear == 0 => True.

∴ front = rear = TmpCell.

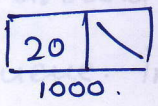
(ie) front = 500

rear = 500



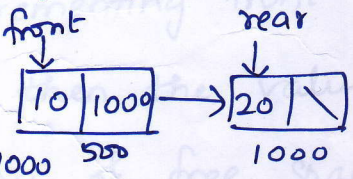
(ii) Enqueue (20)

TmpCell = 1000



rear == 0 => false

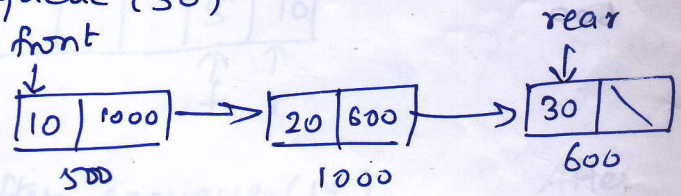
else.



rear -> next = 1000

rear = 1000

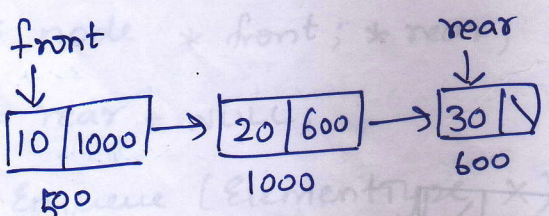
(iii) Enqueue (30)



(iii) Dequeue - It is used to remove an element from the front of the queue.

```
void dequeue()
{
  if (front == NULL)
    printf("Queue Underflow");
  else
  {
    temp = front;
    if (front == rear)
    {
      front = NULL;
      rear = NULL;
    }
    else
      front = front -> next;
    free(temp);
  }
}
```

eg.



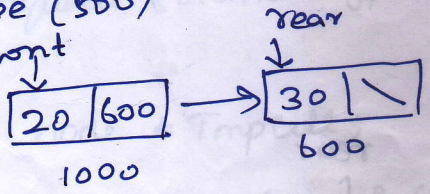
Dequeue():

temp = 500

if (front == rear) (ie) (500 == 600) => false

else front = 1000

free (500)



Dequeue().

temp = 1000

if (1000 == 600) => false

front = 600