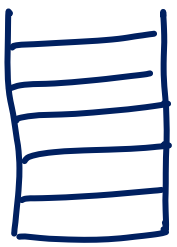
STACK

03/11/2021

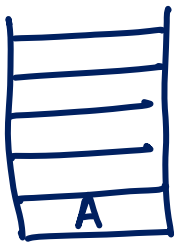
Stack is defined as a list of items in which insertion / deletion can happen only at the top position.

Basically, the operations are best described as LIFO (Last-In-First-Out)

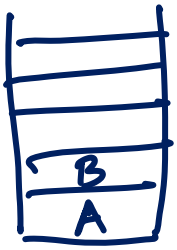
1) Insertion called "Push"
2) Deletion called "Pop"



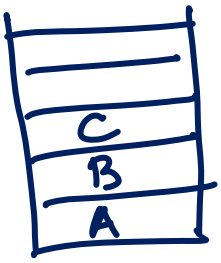
(a) Empty



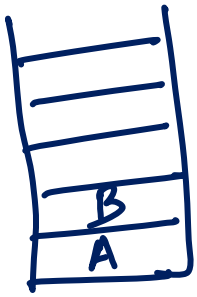
(b) Push A



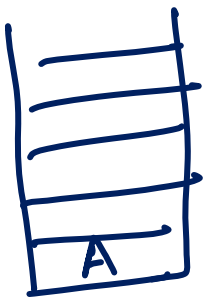
(c) Push B



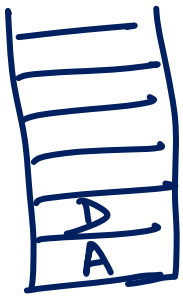
(d) Push C



(e) Pop



(f) Pop



(g) Push D

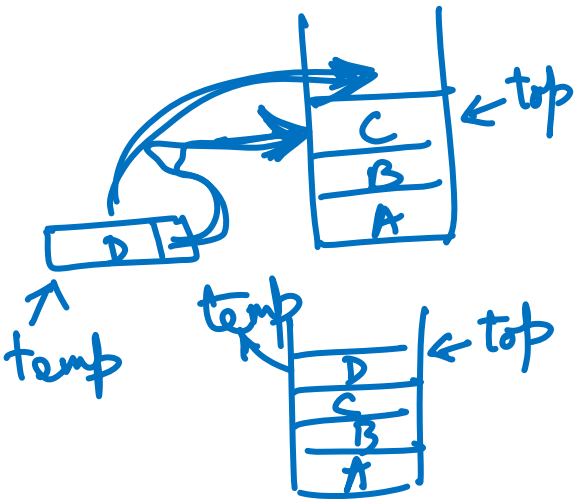
(h) Pop

Linked list implementation of Stack :

```
struct node
{   int info,
    struct node * next ;
};
```

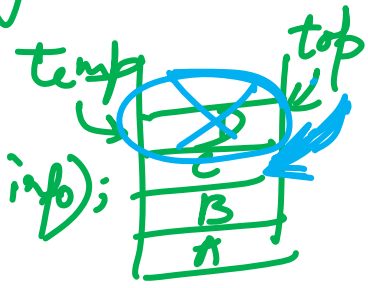
A. Push operation on Stack

```
temp = (struct node *) malloc(sizeof (struct node));
temp → info = pushed_item;
temp → next = top;
top = temp,
```



B. Pop operation on Stack.

```
if (top == NULL)
    printf (" Stack is empty ");
else {  temp = top ;
printf("Popped item is %d", temp→info);
top = top → next ;
free(temp);
}
```



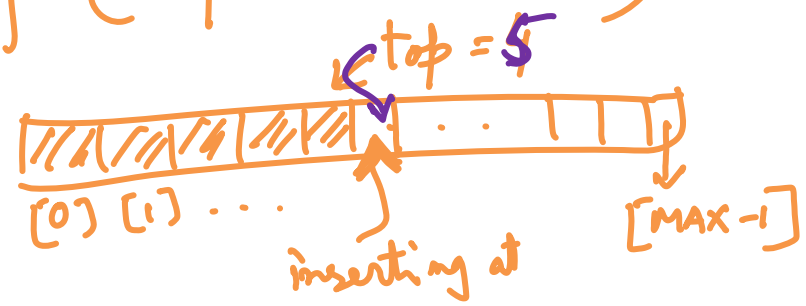
Array Implementation of Stack

A. Push operation

```
#define MAX 20;
.....
main()
{ int arr[MAX];
```

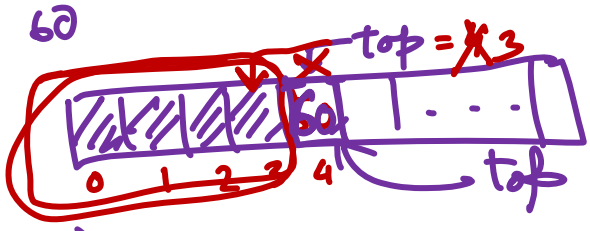
top = -1 for an empty array i.e. empty stack

~~if (top == (MAX-1))~~




```
if (top == (MAX-1))
    printf("Stack Overflow");

else
{ top = top + 1;
  arr[top] = pushed_item;
}
```



B. Pop operation

```
if (top == -1)
    printf("Stack Underflow");

else { printf("Popped Element is %d",
50                arr[top]);
       top = top - 1;
}
```

Array	vs	Linked List
(a) Static Memory allocation		(a) Dynamic Memory allocation
(b) Fixed sized memory declared at the beginning of any program remains allocated even if the user doesn't need it.		(b) The memory locations are allocated run-time as and when needed.
(c) Wastage of valuable memory space can happen.		(c) doesn't happen.

—×—

QUEUE

Dated 04|11|2021

First - In - First - Out Data Structure.
(FIFO)

Examples : Queue of people,
Queue of Cars,
Queue of jobs in a printer
etc.

A. Linked list implementation

```
struct node
    {
        int data;
        struct node * link;
    };
struct node  *temp;
struct node  * front;
struct node  * rear;
```

Insertion within Queue :

```
temp = (struct node *) malloc (sizeof (struct node));
temp -> data = added_item;
temp -> link = NULL;
  if (front == NULL)
            front = temp;
  else
        rear -> link = temp;
rear = temp;
```

Deletion from a Queue :

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

```
else
   {
      temp = front;
      printf("Deleted item is %d ", temp->data);
      front = front->link;
      free (temp);
   }
```

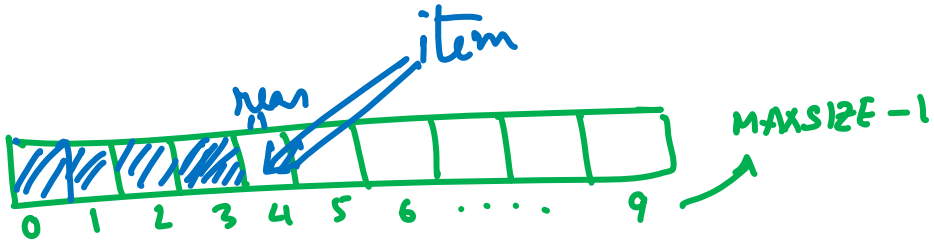
B. Array Implementation

```
#define MAXSIZE 10;
main()
{  int front, rear;
   int queue_arr[MAXSIZE];
   front = rear = -1; /* initial condition */
   int item;
   printf(" enter item to be inserted ");
   scanf("%d", &item);
   :
   :
}
```

Insertion with Queue

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

else
{   if (front == -1)
        front = 0;
    rear = rear + 1;
    queue_arr[rear] = item;
}
```



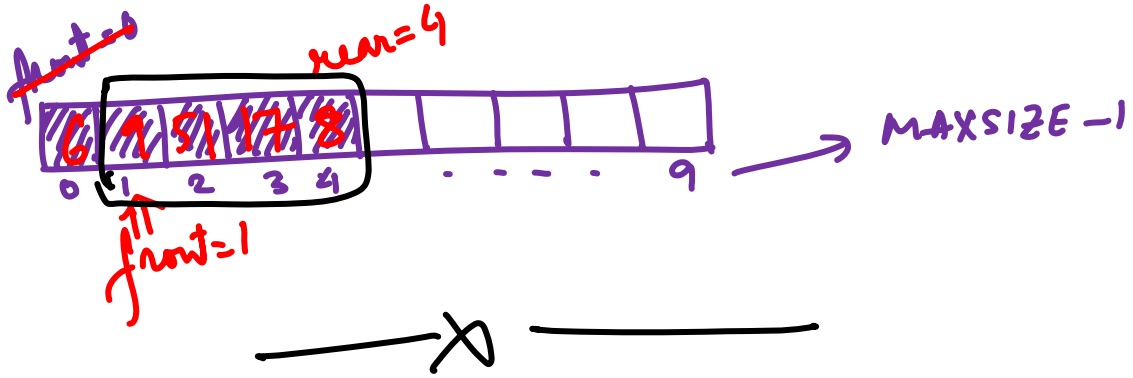
front = 0
rear = 3 + 1 = 4
queue_arr[4] = item;

Deletion operation within Queue

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

else
{   printf("Item to be deleted is %d",
            queue_arr[front]);
    front = front + 1;
}
```

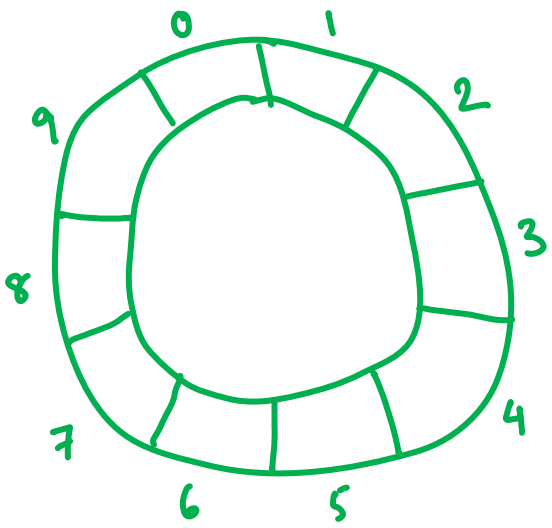
6 ←

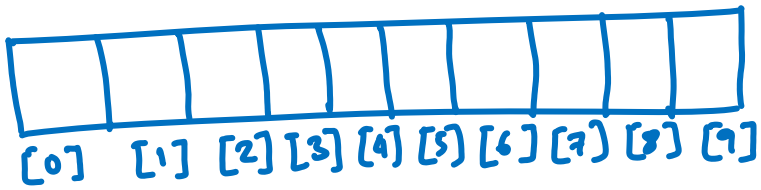


CIRCULAR QUEUE

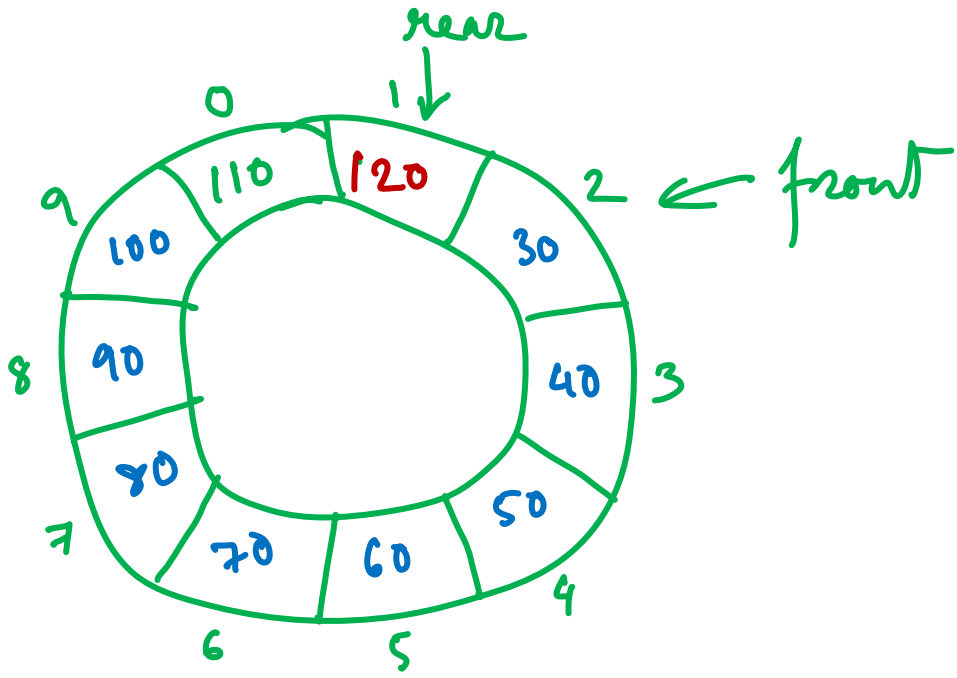
Dated 15/11/2021

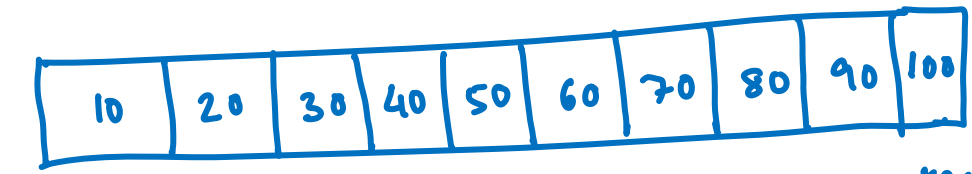
It follows the basic concept of Queue i.e. the FIFO property.
But, it is usually implemented with static or fixed sized memory.





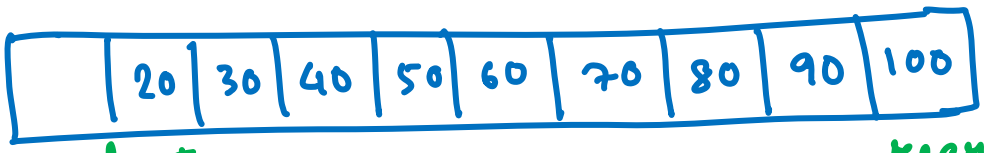

```
# include <stdio.h>
# define MAXSIZE 10 ,
main()
{
    int front, rear ;
    int cqueue_arr [MAXSIZE] ,
    int item ;
    front = rear = -1 ,
    printf(" Enter the value of initial item");
    scanf ("%d", & item);
    :
```



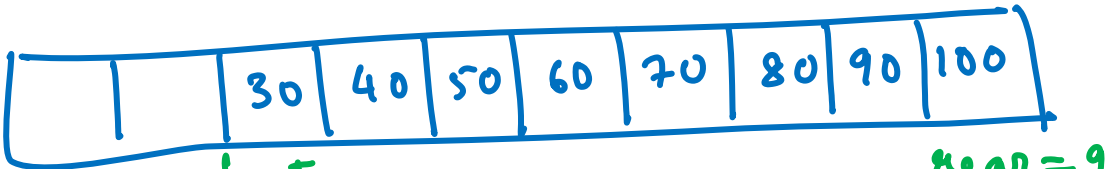


(a) Circular Queue is Full

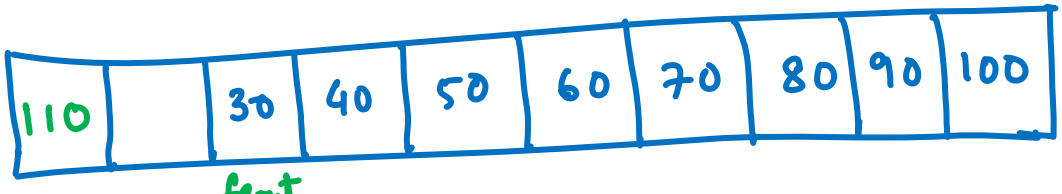
↳ (MAXSIZE-1)



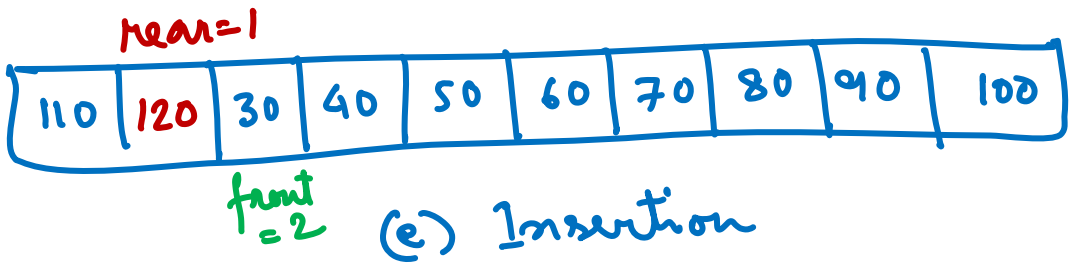
(b) Deletion



(c) Deletion



(d) Insertion



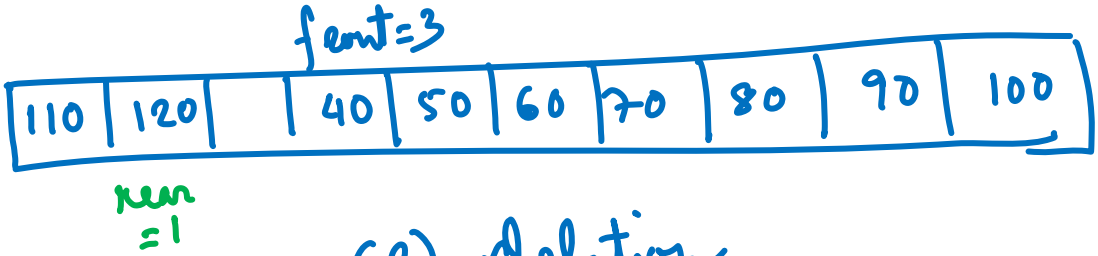
(e) Insertion

After this above insertion, the circular queue is Full again and we observe that "front = rear + 1"

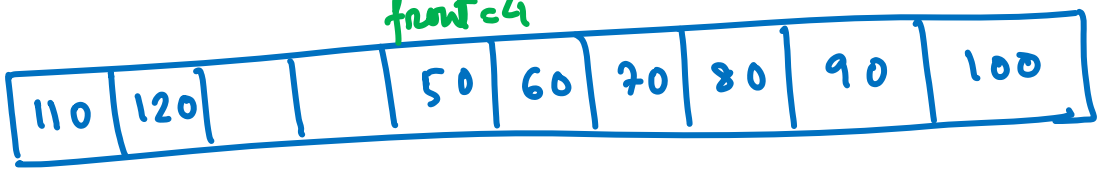
Conditions for the Circular Queue to be Full are — either one of these.

(i) front = 0, rear = MAXSIZE - 1 ;

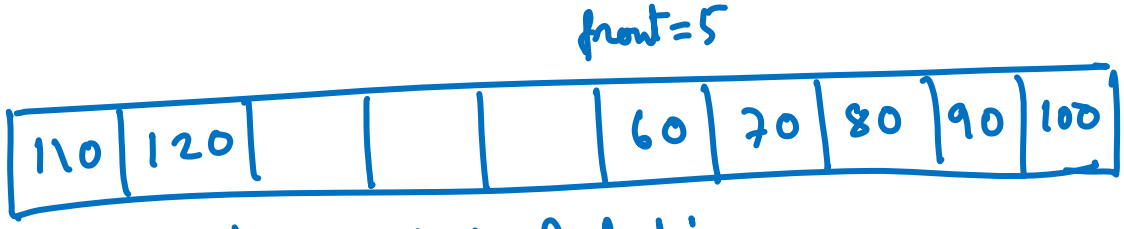
(ii) front = rear + 1 ;



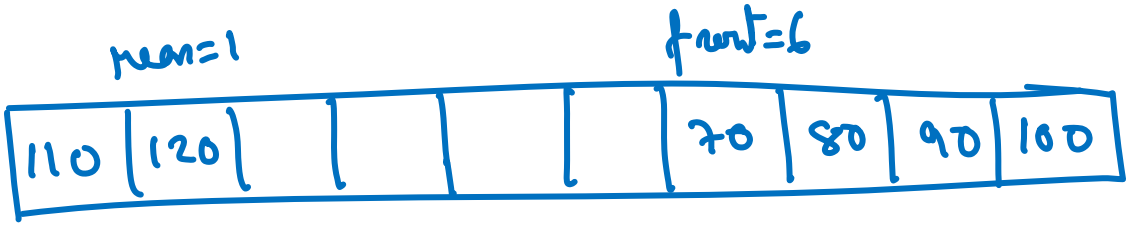
(f) Deletion



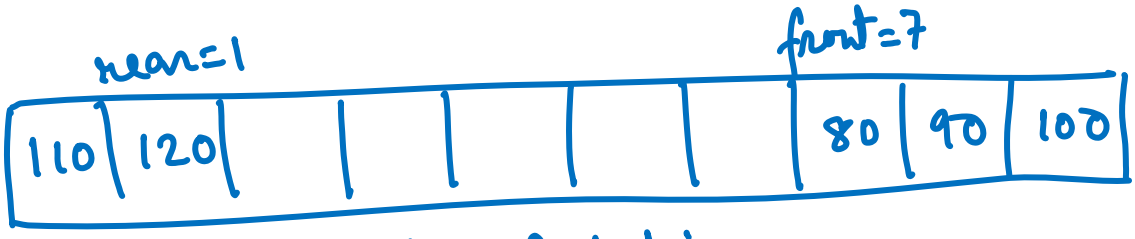
(g) Deletion



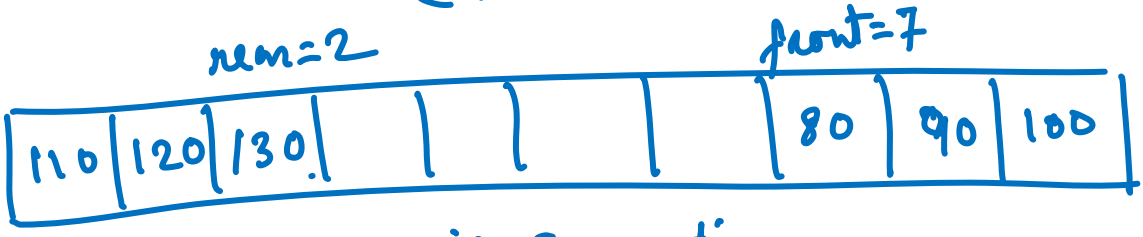
(h) Deletion



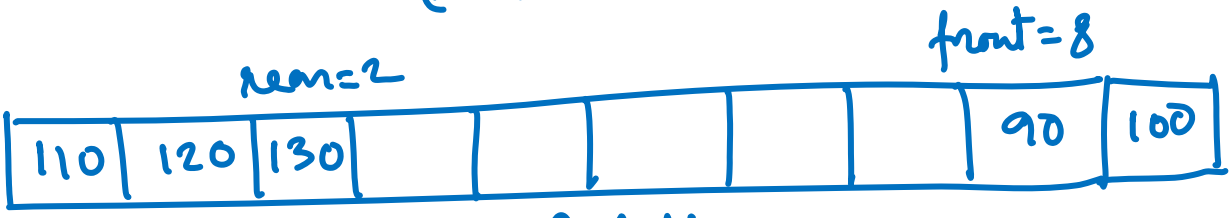
(g) Deletion



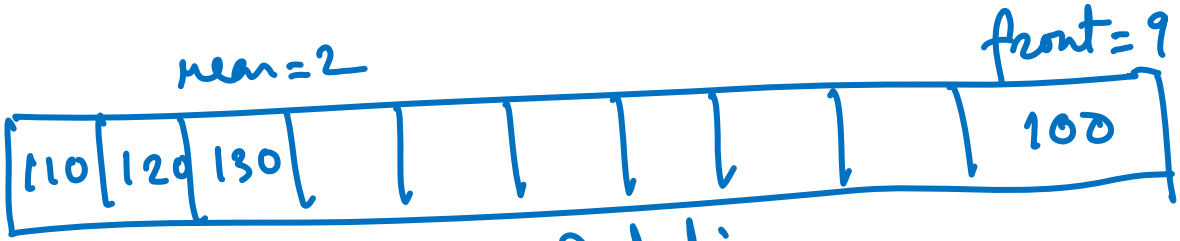
(h) Deletion



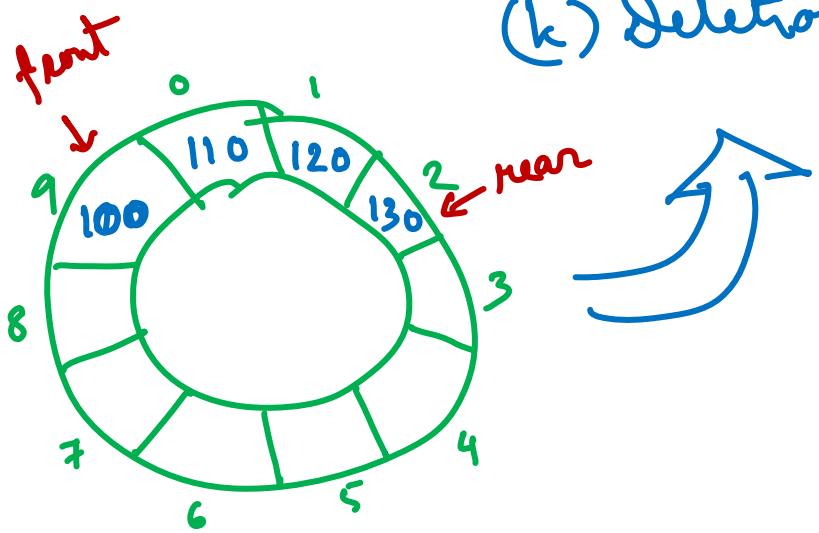
(i) Insertion

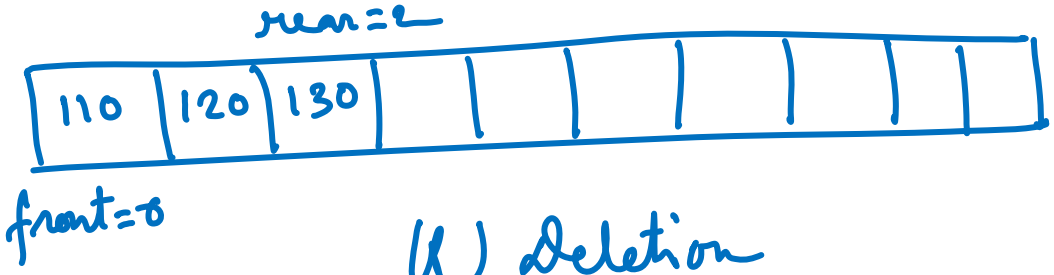


(j) Deletion

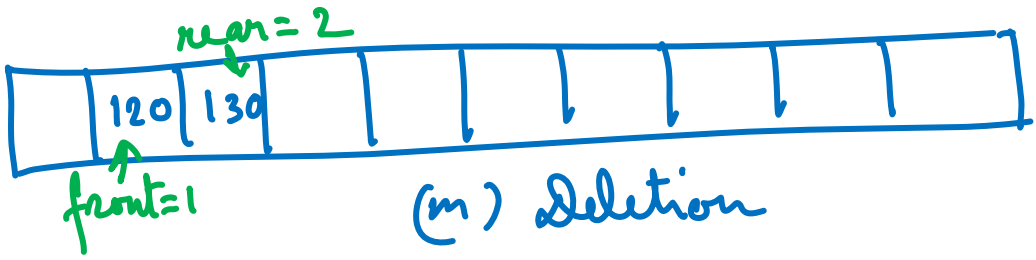


(k) Deletion

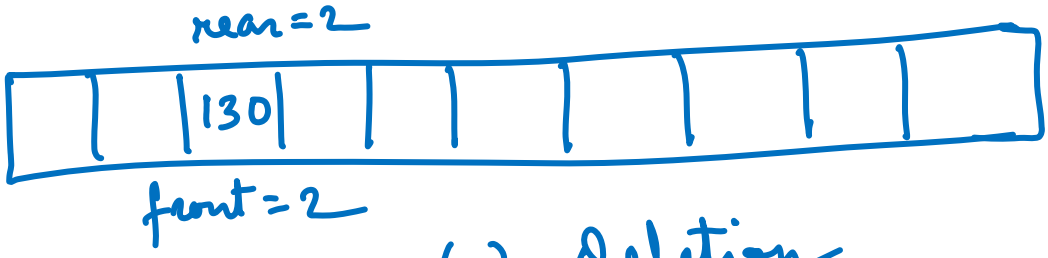




(1) Deletion

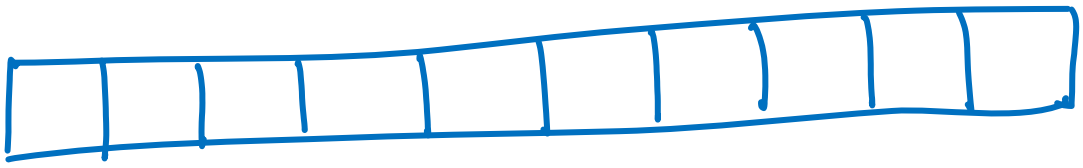


(m) Deletion



(n) Deletion

When the Circular queue has only one element $\boxed{front = rear \neq -1}$



front = -1
rear = -1

(0) Deletion

The condition of an empty Queue is

$\boxed{front = rear = -1}$