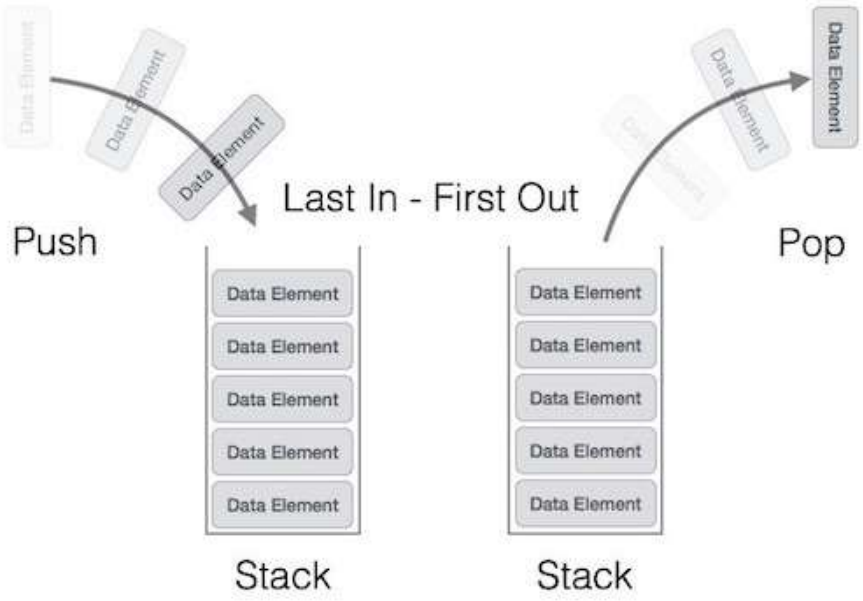
Lab 05: Implementation of Stack and Queue operations

Stack:

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out).



Basic Operations:

Mainly the following three basic operations are performed in the stack:

Push(): Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.

Pop(): Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

Peek() or Top(): Returns top element of stack.

isEmpty(): Returns true if stack is empty, else false.

isFull(): check if stack is full

ALGORITHMS

Before developing algorithms for stack operations, we must decide on stack implementation i.e., we should first answer the question how to represent a stack in the computer‟s memory.

Let the stack be represented by a linear array STACK. An integer variable TOS is used to hold the array-index of the element last inserted onto the stack, i.e., it is the top of stack. An empty stack is represented

by TOS = -1. Another variable MAXSTK is maintained to indicate the maximum number of elements that can be held by the stack. Thus array implementation of stack requires an array to hold stack elements, an integer variable to point to top of stack and another integer variable to indicate the stack capacity.

Algorithm A1: PUSH (STACK, TOS, MAXSTK, ITEM)

This algorithm pushes an ITEM onto a stack.

1. /* Check for OVERFLOW. An overflow is said to occur when an attempt is made to insert onto a stack when it already contains maximum number of elements */ if (TOS = = MAXSTK – 1) then PRINT "OVERFLOW" and return.
2. Set TOS = TOS + 1.
3. Set STACK[TOS] = ITEM.
4. Return.

Algorithm A2: POP (STACK, TOS, ITEM)

This algorithm deletes the top element of STACK and assigns it to the variable ITEM

1. /* Check for UNDERFLOW. An underflow is said to occur when an attempt is made to delete from an empty stack */
   if (TOS = = -1) then PRINT "UNDERFLOW" and return.
2. Set ITEM = STACK[TOS].
3. Set TOS = TOS –1.
4. Return.

. .

Queue:

Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Queue

Basic Operations:

Mainly the following four basic operations are performed on queue:

enqueue() − add (store) an item to the queue.

dequeue() − remove (access) an item from the queue.

peek() − Gets the element at the front of the queue without removing it.

isfull() − Checks if the queue is full.

isempty() − Checks if the queue is empty.

1. Insert Operation:

ALGORITHM1: Addition into a queue(enqueue)

```
procedure addq (item : items);
{add item to the queue q}
begin
   if rear=n
         then queue-full
   else begin
       rear :=rear+1;
       q[rear]:=item;
   end;
end;{of addq}
```

2. Delete Operation:

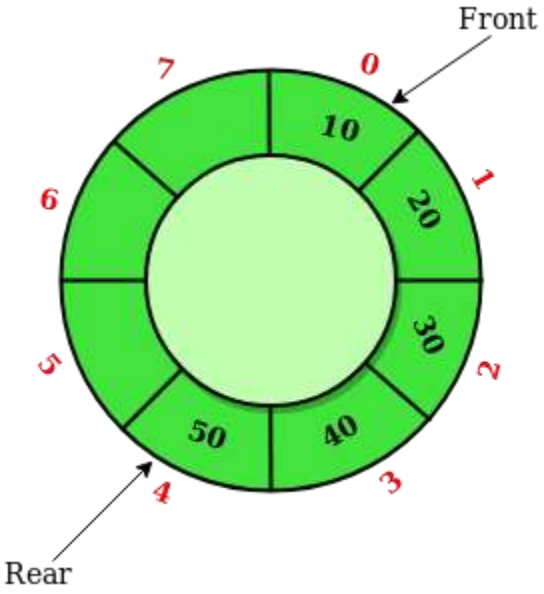
ALGORITHM2: Deletion in a queue (dequeue)

```
procedure deleteq (var item : items);
{delete from the front of q and put into
item}
begin
   if front = rear
         then queue-empty
   else begin
       front := front+1
       item := q[front];
   end;
end; {of deleteq}
```

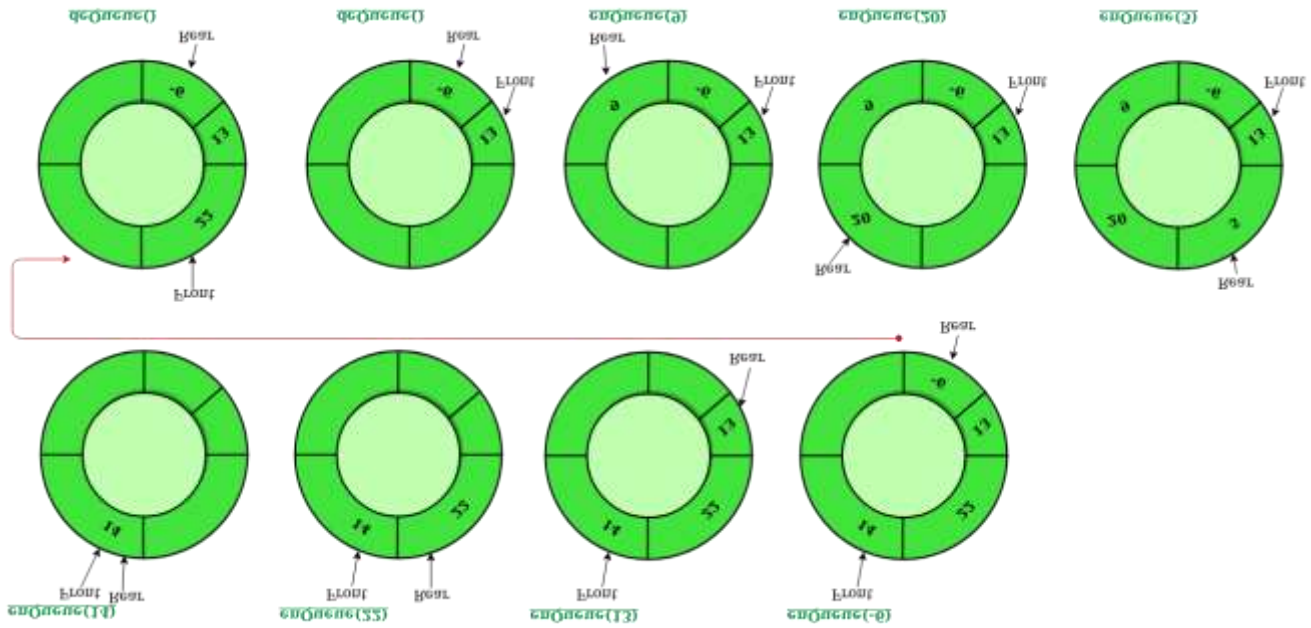
CIRCULAR QUEUE:

Let we have an array Q that contains n elements in which Q[1] comes after Q[n] in the array. When this technique is used to construct a queue then the queue is called circular queue. In other word we can say that a queue is called circular when the last element comes just before the first element.

Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'**.



In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.



OPERATIONS AND ALGORITHMS OF CIRCULAR QUEUE:

1. Insert Operation:

Step 1: [Check for the Overflow]
    If front = 1 and rear = n || rear = front-1
    Output "Overflow" and return
Step 2: [Insert element in the queue.]
    Else if front = 0
    front = 1
    rear = 1
    Q [rear] = value
Step 3: [Check if the rear at the end of the queue]
    Else if rear = n
    rear = 1
    Q [rear] = value
Step 4: [insert the element]
    Else
    Rear = rear + 1
    Q [rear] = value

Step 5: return

2. Delete Operation:

```
Step 1:          [Check for underflow]
    If front = 0
    Output "Underflow" and return
Step 2: [Remove the element]
    Value = Q [front]
Step 3: [Check whether the queue is empty or not]
    If front = rear
    front = 0
    rear = 0
Step 4: [Check the front pointer position]
    Else if front = n
    front = 1
    Else
    front = front + 1
Step 5: [Return the removed element]
    Return [value]
```