Implementation of Stack Using Linked List

Stack Operations

- push(x)
- Pop()
- peek()[top element]
- display()

principle of stack is LIFO(LAST IN FIRST OUT)

- Time complexity of stack is O(1)

In [5]:

```
# creating a node in linked list
class Node:
    # constructor __init__()
    def __init__(self,data):
        self.data=data
        self.ref=None
class StackLL:
    # top is default None
    def __init__(self):
        self.top=None
    # Check if the stack is empty or no
    def isempty(self):
        if self.top is None:
            return True
        else:
            return False
    # To add Nodes into the stack(Push(x))
    def push(self,data):
        if self.top is None:
            self.top=Node(data) # top is pointing to new node if stack is empty
        else:
            newnode=Node(data) # Creating a new node
            newnode.ref=self.top # newnode ref is pointing to top ref
            self.top=newnode # top is now pinting to newnode
    # Removing Nodes from the stack (pop())
    def pop(self):
        if self.isempty():
            return None
        else:
            # removing the top node and makes the preceeding one, the new node
            p=self.top # storing top variable in temporary varaible p
            self.top=self.top.ref # we can create a link from top to second node
            p.ref=None # first node will be deleted
            return p.data # deleted element
    # Display the top element
    def peek(self):
        if self.isempty():
            return None
        else:
            return self.top.data
    # Display the Nodes present in the Stack
    def display(self):
        t=self.top
        if self.isempty():
            print("Stack is Underflow!!!!")
        else:
            while t is not None:
                print(t.data,"--->",end=" ")
                t=t.ref
            return
sll=StackLL() # creating an instance to the class StackLL(object creation)
print("It is empty or Not???",sll.isempty())
sll.push(100)
sll.push(200)
sll.push(300)
sll.push(400)
# display the elements
print(sll.display())
# display top element
```

```
print("Top elemnt is...",sll.peek())
print("After Deletion....")
print(sll.pop())
print(sll.display())
print("Top elemnt is...",sll.peek())
```

```
It is empty or Not??? True
400 --->
300 --->
200 --->
100 --->
None
Top elemnt is... 400
After Deletion....
400
300 --->
200 --->
100 --->
None
Top elemnt is... 300
```

Implementing Queue using Linked List

principle of Queue is FIFO(First In First Out) Time Complexity is O(1)

Queue Operations

- Enqueue
- Dequeue
- Display

In [5]:

```
# A linked list (LL) node
# to store a queue entry
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None

# A class to represent a queue

# The queue, front stores the front node
# of LL and rear stores the last node of LL
class QueueLL:

    def __init__(self):
        self.front = self.rear = None

    def isEmpty(self):
        return self.front == None

    # Method to add an item to the queue
    def EnQueue(self, item):
        temp = Node(item)

        if self.rear == None:
            self.front = self.rear = temp
            return
        self.rear.next = temp
        self.rear = temp

    # Method to remove an item from queue
    def DeQueue(self):

        if self.isEmpty():
            return
        temp = self.front
        self.front = temp.next

        if(self.front == None):
            self.rear = None

# Driver Code
if __name__== '__main__':
    q = QueueLL()
    q.EnQueue(10)
    q.EnQueue(20)
    print("Queue Front " + str(q.front.data))
    q.DeQueue()
    q.EnQueue(30)
    q.EnQueue(40)
    q.EnQueue(50)
    print("Queue Front " + str(q.front.data))
    print("Queue Rear " + str(q.rear.data))
```

```
Queue Front 10
Queue Front 20
Queue Rear 50
```

In [ ]:

```
```