**Code: -**

class Node:

    def \_\_init\_\_(self, data):

*self*.data = data  *# Data to store in the node*

*self*.prev = None  *# Previous pointer (for doubly linked list)*

*self*.next = None  *# Next pointer (for doubly linked list)*

class Deque:

    def \_\_init\_\_(self):

*self*.front = None  *# Front pointer*

*self*.rear = None   *# Rear pointer*

    def enqueueFront(self, data):

        new\_node = Node(data)

        if not *self*.front:  *# If the deque is empty*

*self*.front = *self*.rear = new\_node  *# Both front and rear point to the new node*

        else:

            new\_node.next = *self*.front  *# Link new node to the front*

*self*.front.prev = new\_node  *# Link front's prev pointer to the new node*

*self*.front = new\_node  *# Move front pointer to the new node*

    def enqueueRear(self, data):

        new\_node = Node(data)

        if not *self*.rear:  *# If the deque is empty*

*self*.front = *self*.rear = new\_node  *# Both front and rear point to the new node*

        else:

*self*.rear.next = new\_node  *# Link rear's next to the new node*

            new\_node.prev = *self*.rear  *# Link the new node's prev to the rear*

*self*.rear = new\_node  *# Move rear pointer to the new node*

    def dequeueFront(self):

        if not *self*.front:  *# If the deque is empty*

            print("Deque is empty, cannot dequeue from front.")

            return

        if *self*.front == *self*.rear:  *# If there is only one element*

*self*.front = *self*.rear = None  *# Deque is now empty*

        else:

*self*.front = *self*.front.next  *# Move front pointer to the next node*

*self*.front.prev = None  *# Set the new front's previous pointer to None*

    def dequeueRear(self):

        if not *self*.rear:  *# If the deque is empty*

            print("Deque is empty, cannot dequeue from rear.")

            return

        if *self*.front == *self*.rear:  *# If there is only one element*

*self*.front = *self*.rear = None  *# Deque is now empty*

        else:

*self*.rear = *self*.rear.prev  *# Move rear pointer to the previous node*

*self*.rear.next = None  *# Set the new rear's next pointer to None*

    def display(self):

        if not *self*.front:  *# If the deque is empty*

            print("Deque is empty.")

            return

        current = *self*.front

        while current:

            print(current.data, end=" < - > ")

            current = current.next

        print("None")  *# Indicating the end of the deque*

*# Example usage*

if \_\_name\_\_ == "\_\_main\_\_":

    deque = Deque()

    deque.enqueueFront(10)

    deque.enqueueFront(20)

    deque.enqueueFront(30)

    print("Deque after enqueueFront operations:")

    deque.display()  *# Expected Output: 30 < - > 20 < - > 10 < - > None*

    deque.enqueueRear(40)

    deque.enqueueRear(50)

    print("Deque after enqueueRear operations:")

    deque.display()  *# Expected Output: 30 < - > 20 < - > 10 < - > 40 < - > 50 < - > None*

    deque.dequeueFront()

    print("Deque after dequeueFront operation:")

    deque.display()  *# Expected Output: 20 < - > 10 < - > 40 < - > 50 < - > None*

    deque.dequeueRear()

    print("Deque after dequeueRear operation:")

    deque.display()  *# Expected Output: 20 < - > 10 < - > 40 < - > None*

**OUTPUT:**

