Practical No 4

Aim : Write A Program To Implement Double Linked List

```
class Node:
    def __init__(self,data,nextNode=None,prevNode=None):
        self.data = data
        self.nextNode = nextNode
        self.prevNode = prevNode

class DoubleLinkedList:

    def __init__(self):
        self.head = None
        self.tail = None

    def __grab(self,key):
        target = self.head
        while target :
            if target.data == key:
                return target
            target = target.nextNode

        return None

    def push(self,data):
        newNode = Node(data)
        newNode.nextNode = self.head

        if self.head is not None:
            self.head.prevNode = newNode

        if self.head is None:
            self.tail = newNode

        self.head = newNode

        return True

    def insertBefore(self,key,data):
        curr = self.head

        # To Handling Base Case
        if curr.data == key:
            self.push(data)
            return True

        # To Find The Target Node
        while curr:
            if curr.data == key:
```

```
            target = curr
            break
        curr = curr.nextNode

    # To Check The Target
    if target is None:
        print("The Previous Node Cannot Be Null.")
        return False

    # Creating New Node
    newNode = Node(data)

    # Setting Next Node
    if target.prevNode is not None:
        target.prevNode.nextNode = newNode
    newNode.nextNode = target

    # Setting Previous Node
    newNode.prevNode = target.prevNode
    target.prevNode = newNode

    return True

def insertAfter(self,key,data):
    curr = self.head

    # To Handling Base Case
    if curr.data == key:
        self.push(data)
        return True

    # To Find The Target Node
    target = self.__grab(key)

    # To Check The Target
    if target is None:
        print("The Previous Node Cannot Be Null.")
        return False

    # Creating New Node
    newNode = Node(data)

    # Setting Next Node
    newNode.nextNode = target.nextNode
    target.nextNode = newNode

    # Setting Previous Node
    if newNode.nextNode is not None:
        newNode.nextNode.prevNode = newNode

    if newNode.nextNode is None:
        self.tail = newNode
    newNode.prevNode = target
```

```
        return True

    # Append Function
    def append(self,data):
        # 1. Using Existing Function
        #self.insertAfter(self.tail.data,data)
        #return True

        # 2. Using Tail

        # Creating New Node
        newNode = Node(data)

        # Setting Node
        self.tail.nextNode = newNode
        newNode.prevNode = self.tail

        self.tail = newNode

        return True

    def printList(self):
        curr = self.head

        while curr:
            print(curr.data,end=" --> ")
            curr = curr.nextNode
        print("None")

    def printReverse(self):
        last = self.tail

        # Printing In Reverse
        print("Printing The Element In Reverse : ")
        while last :
            print(last.data,end=" --> ")
            last = last.prevNode
        print("None")

    def delete(self,key):
        curr = self.head
        target = None

        #Finding The Target
        while curr:
            if curr.data == key:
                target = curr
                break
            curr = curr.nextNode

        # If Element Is Not Prasent In The List
        if target is None:
```

```
            print("Element Not Found In The List")
            return False

        # Deleting The Element
        target.prevNode.nextNode = target.nextNode
        target.nextNode.prevNode = target.prevNode
        target = None

        return True

#  Creating The List And Adding Elements
l = DoubleLinkedList()
l.push(1)
l.push(3)
l.push(4)
l.push(4)
l.push(5)

# Inserting Element
print("Before Inserting Element")
l.printList()
l.insertAfter(1,0)
l.insertBefore(1,2)
print("After Inserting Element")
l.printList()

# Checking Funtion If It Works For Base Case
l.insertBefore(5,6)
l.printList()

# Deleting An Element
l.delete(4)
l.printList()

# Printing Element in Reverse
l.printReverse()

#  Appending Element
l.append(-1)
l.printList()
```

```
Before Inserting Element
5 --> 4 --> 4 --> 3 --> 1 --> None
After Inserting Element
5 --> 4 --> 4 --> 3 --> 2 --> 1 --> 0 --> None
6 --> 5 --> 4 --> 4 --> 3 --> 2 --> 1 --> 0 --> None
6 --> 5 --> 4 --> 3 --> 2 --> 1 --> 0 --> None
Printing The Element In Reverse :
0 --> 1 --> 2 --> 3 --> 4 --> 5 --> 6 --> None
6 --> 5 --> 4 --> 3 --> 2 --> 1 --> 0 --> -1 --> None
```