LINKED LIST

```
class Node:
    def __init__(self, data):
        self.data = data  # Data stored in the node
        self.prev = None  # Previous node
        self.next = None  # Next node

class DLL:
    def __init__(self):
        self.head = None  # an initial empty set is formed

    # INSERTION

    def i_beg(self, data):  # Inserts a new node at the start of the list.
        n = Node(data)      #n is the new node of the list
        if self.head != None:
            n.next = self.head
            self.head.prev = n
            self.head = n
        else:
            self.head = n


    def i_end(self, data):  # Inserts a new node at the end of the list.
        n = Node(data)
        if self.head != None:
            k = self.head    #k is the temperory list
            while k.next:
                k = k.next
            k.next = n
            n.prev = k
        else:
            self.head = n


    def i_pos(self, data, p):   # Inserts a node at specific position(p)
        if p!= 1:
            n = Node(data)
            k = self.head
            for _ in range(p - 2):
                if k == None:
                    raise IndexError("Position is outside the range")
                k = k.next
            n.next = k.next
            if k.next:
                k.next.prev = n
            k.next = n
            n.prev = k
```

```
    else:
        self.i_beg(data)


#DELETION

def d_beg(self):     # Deletes the first node
    if self.head == None:
        print("Empty list is found")
        return
    elif self.head.next == None:
        self.head = None
    else:
        self.head = self.head.next
        self.head.prev = None


def d_end(self):      # Deletes the last node
    if self.head is None:
        print("Empty list is found")
        return
    elif self.head.next == None:
        self.head = None
    else:
        k = self.head
        while k.next:
            k = k.next
        k.prev.next = None


def d_pos(self, p):    # Deletes node at specific position.
    if self.head == None:
        print("Empty list is found")
        return
    elif p == 1:
        self.d_beg()
    else:
        k = self.head
        for _ in range(p - 2):
            if k == None or k.next == None:
                raise IndexError("Position is outside the range")
            k = k.next
        if k.next.next:
            k.next = k.next.next
            k.next.prev = k
        else:
            k.next = None

#TRAVERSAL
```

```
    def t_for(self):
        # Traverses the list from head to tail (i.e;forward transverse).
        if self.head != None:
            k = self.head
            while k:
                print(k.data, end=' ')
                k = k.next
            print()
        else:
            print("Empty list is found")



    def t_back(self):
        # Traverses the list from tail to head (i.e;backward transverse)
        if self.head != None:
            k = self.head
            while k.next:
                k = k.next
            while k:
                print(k.data, end=' ')
                k = k.prev
            print()
        else:
            print("Empty list is found")

dll = DLL()

# Inserting elements
dll.i_end(int(input("Insert end value: ")))
dll.i_end(int(input("Insert end value: ")))
dll.i_end(int(input("Insert end value: ")))
dll.i_beg(int(input("Insert beginning value: ")))
dll.i_pos(int(input("Insert value:")), int(input("specific position: ")))

# Traversal
print("Forward Traversal:")
dll.t_for()
print("Backward Traversal:")
dll.t_back()

# Deleting elements
dll.d_beg()
dll.d_pos(int(input("Required deletion: ")))
dll.d_end()

# Traversal after deletion
print("After deletion, Forward Traversal:")
dll.t_for()
```

```
print("After deletion, Backward Traversal:")
dll.t_back()
```