```
class SinglyLinkedList :
    class Node :
        def __init__(self,data,next = None) :
            self.data = data
            if next is None :
                self.next = None
            else :
                self.next = next

    def __init__(self):
            self.header = self.Node(None,None)
            self.size = 0

    def __str__(self):
        s = 'linked data : '
        p = self.header.next
        while p != None :
            s += str(p.data) + ' '
            p = p.next
        return s

    def __len__(self) :
        return self.size

    def isEmpty(self) :
        return self.size == 0

    def indexOf(self,data) :
        q = self.header.next
        for i in range(len(self)) :
            if q.data == data :
                return i
            q = q.next
        return -1

    def isIn(self,data) :
        return self.indexOf(data) >= 0

    def nodeAt(self,i) :
        p = self.header
        for j in range(-1,i) :
            p = p.next
        return p

    def append(self,data) :
        return self.insertAfter(len(self),data)
```

```
def insertAfter(self,i,data) :
    p = self.nodeAt(i-1)
    p.next = self.Node(data,p.next)
    self.size += 1


def deleteAfter(self,i) :
    p = self.nodeAt(i)
    p.next = p.next.next
    self.size -= 1

def removeData(self,data) :
    if self.isIn(data) :
        self.deleteAfter(self.indexOf(data)-1)
```

```
class DoublyLinkedList :
    class Node :
        def __init__(self,data,prev = None,next = None) :
            self.data = data
            if prev is None :
                self.prev = None
            else :
                self.prev = prev
            if next is None :
                self.next = None
            else :
                self.next = next

    def __init__(self):
            self.header = self.Node(None,None,None)
            self.header.next = self.header.prev = self.header
            self.size = 0

    def __str__(self):
        s = 'linked data : '
        p = self.header.next
        for i in range(len(self)) :
            s += str(p.data) + ' '
            p = p.next
        return s

    def __len__(self) :
        return self.size

    def isEmpty(self) :
        return self.size == 0

    def indexOf(self,data) :
        q = self.header.next
        for i in range(len(self)) :
            if q.data == data :
                return i
            q = q.next
        return -1

    def isIn(self,data) :
        return self.indexOf(data) >= 0

    def nodeAt(self,i) :
        p = self.header
        for j in range(-1,i) :
            p = p.next
```

```
    return p

def insertBefore(self,q,data) :
    p = q.prev
    x = self.Node(data,p,q)
    p.next = q.prev = x
    self.size += 1


def append(self,data) :
    self.insertBefore(self.nodeAt(len(self)),data)

def add(self,i,data) :
    self.insertBefore(self.nodeAt(i),data)

def removeNode(self,q) :
    p = q.prev
    x = q.next
    p.next = x
    x.prev = p
    self.size -= 1

def delete(self,i) :
    removeNode(nodeAi(i))

def remove(self,data) :
    q = self.header.next
    while q != self.header :
        if q.data == data:
            self.removeNode(q)
            break
        q = q.next
```