WK06\doublyLinkedList.py

```python
class MyNode:
    def __init__(self, _data):
        self.data = _data
        self.next = None
        self.prev = None

class MyDoublyLinkedList:
    def __init__(self, _data):
        self.head = MyNode(_data)
        self.tail = self.head
        if _data is None:
            self.length = 0
        else:
            self.length = 1

    def append(self, _data):
        if self.head == None:
            self.head = MyNode(_data)
            self.tail = self.head
            return
        else:
            self.tail.next = MyNode(_data)
            self.tail.next.prev = self.tail
            self.tail = self.tail.next
        self.length += 1

    def dis(self): # display
        if self.head is None:
            print('DLL[]')
            return
        node = self.head
        print('DLL[' + str(node.data), end='')
        if node.next is None:
            print(']')
        while node.next is not None:
            node = node.next
            if node.next is None:
                print(f', {str(node.data)}', end=']\n')
            else:
                print(f', {str(node.data)}', end='')

    def forwardSearch(self, _data):
        currentNode = self.head
        while currentNode is not None:
            if currentNode.data == _data:
                return currentNode
            currentNode = currentNode.next
        return None
```

```
49
50      def remove(self, _data):
51          currentNode = self.forwardSearch(_data)
52          if currentNode is None: # data searched for not found
53              print(f'{_data} not found, remove operation failed.')
54              return
55          elif currentNode.next is None: # handles edge case: tail
56              currentNode.prev.next = None
57              self.tail = currentNode.prev
58              del currentNode
59
60          elif currentNode.prev is None: # handles edge case: head
61              currentNode.next.prev = None
62              self.head = currentNode.next
63              del currentNode
64
65          else: # handles removal of any other position within the linked list
66              currentNode.prev.next = currentNode.next
67              currentNode.next.prev = currentNode.prev
68              del currentNode
69          self.length -= 1
70
71  x = MyDoublyLinkedList(0)
72
73  for i in range(1, 10, 1):
74      x.append(i)
75
76  print('Initial Doubly Linked List')
77  x.dis()
78  print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
79
80  x.remove(0)
81  print('Edge case: head - Remove 0')
82  x.dis()
83  print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
84
85  x.remove(9)
86  print('Edge case: tail - Remove 9')
87  x.dis()
88  print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
89
90  x.remove(5)
91  print('Middle case - Remove 5')
92  x.dis()
93  print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
94
95  print('Data-not-found case - Remove 20')
96  x.remove(20)
97  x.dis()
98  print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
```

```
99
100 print('Append 21')
101 x.append(21)
102 x.dis()
103 print(f'Head: {x.head.data} \tTail: {x.tail.data}  \tLen: {x.length}\n')
```