

WK06\doublyLinkedList.py

```
1 class MyNode:
2     def __init__(self, _data):
3         self.data = _data
4         self.next = None
5         self.prev = None
6
7 class MyLinkedList:
8     def __init__(self, _data):
9         self.head = MyNode(_data)
10        self.tail = self.head
11        if _data is None:
12            self.length = 0
13        else:
14            self.length = 1
15
16    def append(self, _data):
17        if self.head == None:
18            self.head = MyNode(_data)
19            self.tail = self.head
20            return
21        else:
22            self.tail.next = MyNode(_data)
23            self.tail.next.prev = self.tail
24            self.tail = self.tail.next
25            self.length += 1
26
27    def dis(self): # display
28        if self.head is None:
29            print('DLL[]')
30            return
31        node = self.head
32        print('DLL[' + str(node.data), end='')
33        if node.next is None:
34            print(']')
35        while node.next is not None:
36            node = node.next
37            if node.next is None:
38                print(f', {str(node.data)}', end=']\n')
39            else:
40                print(f', {str(node.data)}', end='')
41
42    def forwardSearch(self, _data):
43        currentNode = self.head
44        while currentNode is not None:
45            if currentNode.data == _data:
46                return currentNode
47            currentNode = currentNode.next
48        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             self.length -= 1
69
70 x = MyLinkedList(0)
71
72 for i in range(1, 10, 1):
73     x.append(i)
74
75 print('Initial Doubly Linked List')
76 x.dis()
77 print(f'Head: {x.head.data} \tTail: {x.tail.data} \tLen: {x.length}\n')
78
79 x.remove(0)
80 print('Edge case: head - Remove 0')
81 x.dis()
82 print(f'Head: {x.head.data} \tTail: {x.tail.data} \tLen: {x.length}\n')
83
84 x.remove(9)
85 print('Edge case: tail - Remove 9')
86 x.dis()
87 print(f'Head: {x.head.data} \tTail: {x.tail.data} \tLen: {x.length}\n')
88
89 x.remove(5)
90 print('Middle case - Remove 5')
91 x.dis()
92 print(f'Head: {x.head.data} \tTail: {x.tail.data} \tLen: {x.length}\n')
93
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')
```