Lab-VI

Hasan Amin
(374866)

CS-250
Data Structures and Algorithms

School of Natural Sciences

SLL Class:

Code

```
class Node:
    def __init__(self, value):
        self.data = value
        self.nextnode = None


class SLL:
    def __init__(self, data=None):
        self.head = Node(data)
```

```
    def isEmpty(self):  # Test for empty list
        return self.head is None  # True if & only if no 1st Link
```

```
    def traverse(self, nextnode):   # i give it the next node for the particular node and it returns
me that particular node
        currnode = self.head
        while currnode.nextnode != nextnode:
            currnode = currnode.nextnode
        return currnode
```

```
    def get_previous_node(self, node: Node):
        currnode = self.head
        while currnode.nextnode != node:
            currnode = currnode.nextnode
        return currnode
```

```
    def traverse_by_index(self, index):
        currnode = self.head
        for i in range(index):
            currnode = currnode.nextnode
        return currnode

    def append(self, data):
        new_node = Node(data)
        last_node = self.traverse(None)
        last_node.nextnode = new_node
```

```
    def insert(self, index, data):
        new_node = Node(data)
        if index > self.count() + 1:
            raise ValueError("List index out of range")
        that_node = self.traverse_by_index(index)
        if that_node != self.head:
            prev_node = self.get_previous_node(that_node)
            prev_node.nextnode = new_node
            new_node.nextnode = that_node
        else:
            new_node.nextnode = that_node
            self.head = new_node
```

```
    def deleteFirst(self):
```

```
        if self.isEmpty():  # Empty list? Raise an exception
            raise Exception("Cannot delete first of empty list")
        curr_node = self.head
        self.head = self.head.nextnode
        return curr_node
```

```
    def deleteLast(self):
        curr_node = self.head
        if curr_node == None or curr_node.nextnode == None:
            self.head = None
            return curr_node
```

```
        temp = self.head
        while temp.nextnode is not None:
            current = temp
            temp = temp.nextnode
        current.nextnode = None
        return temp.data
```

```
    def SearchElement(self, val):  # search element in linked list
        print(f"Searching for element {val}")
        if self.isEmpty():
            print("nothing to remove,list is empty")
            return
```

```
        temp = self.head
```

```
        while temp.nextnode is not None:
            if temp.data == val:
                print(f"Value Found {temp.data}")
                return True
            temp = temp.nextnode
        if temp.data == val:
            print(f"\nElement Found {temp.data}")
        else:
            return False
```

```
    def count(self): # count the number of nodes
        currnode = self.head
        counter = 0
        while currnode is not None:
            counter += 1
            currnode = currnode.nextnode
        return counter


    def unique(self):
        if self.isEmpty():
            return  # No duplicates to remove from an empty list
```

```
        current = self.head
```

```
        while current is not None:
            # Compare the current node's data with subsequent nodes
            runner = current
            while runner.nextnode is not None:
                if runner.nextnode.data == current.data:
                    # Remove the duplicate node
                    runner.nextnode = runner.nextnode.nextnode
                else:
```

```
                runner = runner.nextnode
            current = current.nextnode
```

```
    def removeEvens(self):
        sll2 = SLL(self.head.data)
        currnode = self.head
        self.remove(self.head.data)
        while currnode.nextnode.nextnode is not None:
            currnode = currnode.nextnode.nextnode
            sll2.append(currnode.data)
            self.remove(currnode.data)
            if currnode.nextnode is None:
                break
        return sll2
```

```
    def groupOddEvens(self):
        even = self.removeEvens()
        currnode = self.head
        while currnode.nextnode is not None:
            currnode = currnode.nextnode
        currnode.nextnode = even.head
```

```
    def count50s(self):
        c = 0
        currnode = self.head
        while currnode is not None:
            if currnode.data == 50:
                c += 1
            currnode = currnode.nextnode
        return c
```

```
    def remove(self, info):
        currnode = self.head
        if currnode is None:
            raise ValueError(f"{info} not found in list")
```

```
        if currnode.data == info:
            self.head = currnode.nextnode
        else:
            while currnode.nextnode is not None and currnode.nextnode.data != info:
                currnode = currnode.nextnode
```

```
            if currnode.nextnode and currnode.nextnode.data == info:
                currnode.nextnode = currnode.nextnode.nextnode
            else:
                raise ValueError(f"{info} not found in list")

    def pop(self, index): # removing node from a specific location/index
        if self.isEmpty():  # Empty list? Raise an exception
            raise ValueError("List is Empty")
        elif index > len(self):
            raise ValueError("list index out of range")
        else:
            toDelete = self.traverse_by_index(index)
            self.remove(toDelete.data)
```

```
    def destroy(self):
        currnode = self.head
        while currnode is not None:
            temp = currnode.nextnode
            del currnode
```

```
            currnode = temp
        self.head = None  # Set the head to None to indicate an empty list

    def __len__(self) -> int:
        return self.count()
```

```
    def __str__(self):  # Build a string representation
        result = "["  # Enclose list in square brackets
        cur_node = self.head  # Start with first link
```

```
        while cur_node is not None:  # Keep going until no more links
            if len(result) > 1:  # After first link,
                result += " --> "  # separate links with right arrowhead
            result += str(cur_node.data)  # Append string version of link
            cur_node = cur_node.nextnode  # Move on to nextnode link
        return result + "]"  # Close with square bracket
```

```
    def __eq__(self, obj: object) -> bool:
        if (
            len(self) != len(obj)
            or (self is None and obj is not None)
            or (self is not None and obj is None)
        ):
            return False
        curr1 = self.head
        curr2 = obj.head
        while curr1.data is not None and curr2.data is not None:
            if curr1.data != curr2.data:
                return False
            curr2 = curr2.nextnode
            curr1 = curr1.nextnode
        return True
```

```
    def __ne__(self, obj: object) -> bool:
        return not self.__eq__(obj)
```

Class Method Testing:

Code

```
from SinglyLinkedList import SLL

sll=SLL("abc")
```

```
sll.append("abc")
sll.append("abc")
sll.append("aec")
sll.append("axc")
sll.append("ahc")
sll.append(50)
sll.append(50)
sll.append(50)
sll.append(50)
sll.append(234)
sll.insert(0,53)
sll.insert(3,53)
sll.insert(2,100)
```

```
print(f"Linked List: {sll}   Count of 50: {sll.count50s()}")

sll.unique()
print(f"Duplicates removed: {sll}")

sll.groupOddEvens()
print(f"OddEvens Grouped: {sll}")

sll.remove("axc")
print(f'Removed "axc" from my SLL: {sll}')

sll.pop(0)
print(f'Popped the node at 0th index from my SLL: {sll}')

mysll2=SLL()
mysll2=sll.removeEvens()
print(f"Odd Nodes:{sll} Even Nodes: {mysll2}")

mysll2.destroy()
print(f"Destroyed My Even Nodes SLL:{mysll2}")
```

Output

```
Linked List: [53 --> abc --> 100 --> abc --> 53 --> abc --> aec --> axc --> ahc --> 50 --> 50 --> 50 --> 50 --> 234]   Count of 50: 4
Duplicates removed: [53 --> abc --> 100 --> aec --> axc --> ahc --> 50 --> 234]
OddEvens Grouped: [abc --> aec --> ahc --> 234 --> 53 --> 100 --> axc --> 50]
Removed "axc" from my SLL: [abc --> aec --> ahc --> 234 --> 53 --> 100 --> 50]
Popped the node at 0th index from my SLL: [aec --> ahc --> 234 --> 53 --> 100 --> 50]
Odd Nodes:[ahc --> 53 --> 50] Even Nodes: [aec --> 234 --> 100]
Destroyed My Even Nodes SLL:[]
```