CMPSC 462 – In-class Assignment-7 (30 points)
Linked Lists – Singly and Doubly

Due date: 11/08/2022

Note: attach screenshots of your program and results under each programming exercises. Please make sure that the screenshot is readable. Don't attach a very small screenshot image.

Exercise-1: 15 points

Develop a node class and a singly list class. The node class should have two state variables namely data and nextNode. The singly list class should contain the following methods:

- MiddleInsert – insert a node somewhere in the middle of the list
- StartInsert – insert a node at start of the Linked list
- EndInsert – insert a node at the end of the Linked list
- Delete – delete a node
- Traverse – prints all the node's data
- Reverse – reverses the linked list

```
print("Single Link")
a = SinglyList(1)
a.StartInsert(2)
a.StartInsert(5)
a.StartInsert(7)
a.StartInsert(5)
a.Traverse()
a.MiddleInsert(3, 10)
a.Traverse()
a.EndInsert(14)
a.EndInsert(11)
a.Traverse()
a.Delete(4)
a.Traverse()
a.Reverse()
a.Traverse()
```

PROBLEMS OUTPUT DEBUG CONSOLE JUPYTER TERMINAL

```
➜ Assignment-7 python main.py
Single Link
1 -> 5 -> 7 -> 5 -> 2
1 -> 5 -> 7 -> 10 -> 5 -> 2
1 -> 5 -> 7 -> 10 -> 5 -> 2 -> 14 -> 11
1 -> 5 -> 7 -> 10 -> 2 -> 14 -> 11
11 -> 14 -> 2 -> 10 -> 7 -> 5 -> 1
```

```python
class SinglyList(Node):
    def __init__(self, data) -> None:
        super().__init__(data)

    def MiddleInsert(self, index, value):
        new = Node(value)
        node = self
        # traverse the list to the target index
        for _ in range(0, index-1):
            node = node.next_node

        # point the new node to the node that the target index pointed to
        new.next_node = node.next_node
        # point the current node's next to the new node
        node.next_node = new

    def StartInsert(self, value):
        # create a new node and point that new node's next to the current next
        new = Node(value, self.next_node)

        # replace current next to the new node
        self.next_node = new

    def EndInsert(self, value):
        new = Node(value)
        node = self
        # traverse until the end
        while node.next_node:
            node = node.next_node

        # add new node as the end node's next
        node.next_node = new
```

```
    def Delete(self, index):
        node = self
        # go to target index minus 1
        for _ in range(0, index-1):
            node = node.next_node

        # replace the current node's next node to the actual target index's next node
        # which will effectively remove the targeted index node
        node.next_node = node.next_node.next_node

    def Traverse(self):
        # go through each node and print it's data
        node = self
        while node.next_node:
            print(str(node.data) + " -> ", end="")
            node = node.next_node

        # print the final node as the loop won't print it
        print(node.data)

    def Reverse(self):
        # list use to store data of every single nodes
        all_nodes = []
        node = self
        # go through each node and store its data
        while node.next_node:
            all_nodes.append(node.data)
            node = node.next_node
        all_nodes.append(node.data)

        # iterate through all value of all_nodes in reverse and update
        # the value of the nodes
        node = self
        for _ in range(0, len(all_nodes)):
            node.data = all_nodes.pop()
            node = node.next_node
```

```
# real world usage single link
# using it to store adjacent vertices for a graph
# using it to implement a queue
# using it for dynamic memory allocations
```

List out 3 real time usage of Linked List as comments in the program.

Note:

- Choose appropriate inputs for the above functions
- You can refer any source but try to follow the algorithm described in the lecture video
- Write the algorithm for each function as comments in the program
- Test the class's function with an example.
- Attach screenshots of the program and results

Exercise-2: 15 points

Develop a node class and a doubly list class. The node class should have three state variables namely data, prevNode and nextNode. The doubly list class should contain the following methods:

- MiddleInsert – insert a node somewhere in the middle of the list
- StartInsert – insert a node at start of the Linked list
- EndInsert – insert a node at the end of the Linked list
- Delete – delete a node
- Traverse – prints all the node's data
- Reverse – reverses the linked list

```
class NodeDouble():
    def __init__(self, data, next_node:'NodeDouble' = None, prev_node:'NodeDouble' = None) -> None:
        self.data = data
        self.next_node = next_node
        self.prev_node = prev_node

class DoubleLink(NodeDouble):
    def __init__(self, data) -> None:
        super().__init__(data)

    def MiddleInsert(self, index, value):
        new = NodeDouble(value)
        node = self
        # traverse the list to the target index
        for _ in range(0, index-1):
            node = node.next_node

        # point the new node to the node that the target index pointed to
        new.next_node = node.next_node
        # do the same for previous node
        new.prev_node = node
        # point the target index + 1's previous node to index
        new.next_node.prev_node = new
        # point the current node's next to the new node
        node.next_node = new
```

```python
def StartInsert(self, value):
    # create a new node and point that new node's next to the current next, and point it's prev node to current

    next_node = self.next_node if self.next_node is not None else self

    new = NodeDouble(value, next_node, self)
    # replace next node's prev to the new node
    if self.next_node is not None:
        self.next_node.prev_node = new
    # replace current next to the new node
    self.next_node = new

    # if the current head dont have an end add it
    if self.prev_node is None:
        self.prev_node = new

def EndInsert(self, value):
    # if the current head dont have an end add it
    if self.prev_node is None:
        self.prev_node = new
    # create a new node, that point to current node as next and last node as prev
    new = NodeDouble(value, self, self.prev_node)
    # add new node as the end node's next
    self.prev_node.next_node = new
    self.prev_node = new
```

```python
def Delete(self, index):
    node = self
    # go to target index minus 1
    for _ in range(0, index-1):
        node = node.next_node

    # replace the current node's next node to the actual target index's next node
    # replace actual target index's next node's prev node to current node
    # which will effectively remove the targeted index node
    node.next_node.next_node.prev_node = node
    node.next_node = node.next_node.next_node

def Traverse(self):
    # go through each node and print it's data
    node = self
    while True:
        print(str(node.data) + " <-> ", end="")
        node = node.next_node
        if node is self:
            break

    # after the loop the node end up at the starting point, aka HEAD
    print(str(node.data) + " (HEAD)")
```

```python
def Reverse(self):
    node = self
    # function iterate through all element and flip it's prev and next nodes
    while True:
        temp = node.next_node
        node.next_node = node.prev_node
        node.prev_node = temp

        node = node.next_node
        if node is self:
            break
```

```
# double link list real world applications
# using it for undo and redo
# using it for browser's history button of going back and forth
# using it for media player's going to next or previous songs
```

List out 3 real time usage of Doubly Linked List as comments in the program.

Note:

- Choose appropriate inputs for the above functions
- You can refer any source but try to follow the algorithm described in the lecture-4 video (Feb 19th class)
- Write the algorithm for each function as comments in the program
- Test the class's function with an example.
- Attach screenshots of the program and results