CMPSC 132: Programming and Computation II

Lab 4 (10 points) Due date: February 26th, 2022, 11:59 PM EST

Goal: Familiarize with the singly linked list data structure. By completing this lab, you should get a better grasp on how to link items in memory to implement data structures that use linked list representations.

General instructions:

- The work in this assignment must be your own original work and be completed alone.
- The instructor and course assistants are available on Teams and with office hours to answer any questions you may have. You may also share testing code on Teams.
- A doctest is provided to ensure basic functionality and may not be representative of the full range of test cases we will be checking. Further testing is your responsibility.
- Debugging code is also your responsibility.
- You may submit more than once before the deadline; only the latest submission will be graded.

Assignment-specific instructions:

- Download the starter code file from Canvas. Do not change the function names or given starter code in your script.
- You are not allowed to use any other data structures for the purposes of manipulating/sorting elements, nor may you use any modules from the Python to insert and remove elements from the list.
- You are not allowed to swap data from the nodes when adding the node to be sorted. Traversing the linked list and updating 'next' references are the only required and acceptable operations
- You are not allowed to use any kind of built-in sorting method or import any other libraries
- Each question has different requirements, read them carefully and ask questions if you need clarification. No credit is given for code that does not follow directions.
- If you are unable to complete a function, use the pass statement to avoid syntax errors

Recommendations to complete this assignment:

- Watch the lectures first. Before you start each method, work on paper to see what pointers need to be updated based on the position of the Node.
- Make sure you update the head and tail pointers according to the operation performed
- Starter code contains the special methods __str__ and __repr__. Do not modify them, use them to ensure your methods are updating the elements in the list correctly

Submission format:

- Submit your code in a file named LAB4.py file to the Lab 4 Gradescope assignment before the due date.
- As a reminder, code submitted with syntax errors does not receive credit, please run your file before submitting.

Section 1: The SortedLinkedList class (10 points)

In our Hands On – Linked List I and II lecture from Module 5, we worked on the implementation of a Singly Linked List. That data structure keeps the elements of the linked list unsorted. Based on the LinkedList class code completed during the video-lecture, implement the data structure SortedLinkedList with the following methods:

add(self, item) **(3 points**)

adds a new Node with value=item to the list making sure that the **ascending** order is preserved. It needs the item and returns nothing but modifies the linked list. The items in the list might not be unique, but you can assume every value will be numerical (int and float). You are not allowed to traverse the linked list more than once (only one loop required).

Input (excluding self)

int or float	*item*	A numerical value that represents the value of a Node object

Examples:

```
>>> x=SortedLinkedList()
>>> x.add(8.76)
>>> x.add(7)
>>> x.add(3)
>>> x.add(-6)
>>> x.add(58)
>>> x.add(33)
>>> x.add(1)
>>> x.add(-88)
>>> print(x)
Head:Node(-88)
Tail:Node(58)
List:-88 -> -6 -> 1 -> 3 -> 7 -> 8.76 -> 33 -> 58
```

replicate(self) **(2.5 points)**

Returns a new SortedLinkedList object where each element of the linked list appears its node's value number of times (3 -> 1 results in 3 -> 3 -> 3 -> 1). Negative numbers and floats are repeated only once immediately after that node. For 0, the node is added, but not repeated. Method returns None if the list is empty. *Hint*: Using the add method from above could be very useful here!

Output

SortedLinkedList	A new linked list object that repeats values without disturbing the original list
None	None keyword is returned if the original list is empty

Examples:

```
>>> x=SortedLinkedList()
>>> x.add(4)
>>> x.replicate()
Head:Node(4)
Tail:Node(4)
List:4 -> 4 -> 4 -> 4
>>> x.add(-23)
>>> x.add(2)
>>> x.add(1)
>>> x.add(20.8)
>>> x
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> 1 -> 2 -> 4 -> 20.8
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> 1 -> 2 -> 2 -> 4 -> 4 -> 4 -> 4 -> 20.8 -> 20.8
>>> x.add(-1)
>>> x.add(0)
>>> x.add(3)
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> -1 -> -1 -> 0 -> 1 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 4 -> 4 -> 4 ->
20.8 -> 20.8
>>> x
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -1 -> 0 -> 1 -> 2 -> 3 -> 4 -> 20.8
>>> x.add(2)
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> -1 -> -1 -> 0 -> 1 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 4 -> 4
-> 4 -> 20.8 -> 20.8
```

removeDuplicates(self) **(2.5 points)**

Removes any duplicate nodes from the list, so it modifies the original list. This method must traverse the list only once. It modifies the original list.

Output

None	Nothing is returned when the method completes the work

Examples:

```
>>> x=SortedLinkedList()
>>> x.removeDuplicates()
>>> x
Head:None
Tail:None
List:
>>> x.add(1)
>>> x.add(1)
>>> x.add(1)
>>> x.add(1)
>>> x
Head:Node(1)
Tail:Node(1)
List:1 -> 1 -> 1 -> 1
>>> x.removeDuplicates()
>>> x
Head:Node(1)
Tail:Node(1)
List:1
>>> x.add(1)
>>> x.add(2)
>>> x.add(2)
>>> x.add(2)
>>> x.add(3)
>>> x.add(4)
>>> x.add(5)
>>> x.add(5)
>>> x.add(6.7)
>>> x.add(6.7)
>>> x
Head:Node(1)
Tail:Node(6.7)
List:1 -> 1 -> 2 -> 2 -> 2 -> 3 -> 4 -> 5 -> 5 -> 6.7 -> 6.7
>>> x.removeDuplicates()
>>> x
Head:Node(1)
Tail:Node(6.7)
List:1 -> 2 -> 3 -> 4 -> 5 -> 6.7
```

Additional Grading Notes **(2 pts)**

The grading script will perform a series of mixed linked list operations and compare the final status of your list. The last 2 points are based on the correctness of the Linked List after the mixed operations are completed. Verify that all three methods work correctly when method calls are performed in a mixed manner.

Example:

```
>>> x=SortedLinkedList()
>>> x.add(4.5)
>>> x.add(-3)
>>> x.add(0)
>>> x.add(5)
>>> x.add(2)
>>> x.add(-9)
>>> x.add(12.7)
>>> x.add(-3.5)
>>> x.add(2)
>>> x.add(4)
>>> x.add(1)
>>> x.add(3)
>>> x
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -3.5 -> -3 -> 0 -> 1 -> 2 -> 2 -> 3 -> 4 -> 4.5 -> 5 -> 12.7
>>> x.replicate()
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -9 -> -3.5 -> -3.5 -> -3 -> -3 -> 0 -> 1 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 ->
4 -> 4 -> 4 -> 4.5 -> 4.5 -> 5 -> 5 -> 5 -> 5 -> 5 -> 12.7 -> 12.7
>>> x.removeDuplicates()
>>> x
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -3.5 -> -3 -> 0 -> 1 -> 2 -> 3 -> 4 -> 4.5 -> 5 -> 12.7
>>> x.add(0)
>>> x.add(3)
>>> x
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -3.5 -> -3 -> 0 -> 0 -> 1 -> 2 -> 3 -> 3 -> 4 -> 4.5 -> 5 -> 12.7
>>> myList=x.replicate()
>>> myList
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -9 -> -3.5 -> -3.5 -> -3 -> -3 -> 0 -> 0 -> 1 -> 2 -> 2 -> 3 -> 3 -> 3 -> 3 -> 3 ->
3 -> 4 -> 4 -> 4 -> 4 -> 4.5 -> 4.5 -> 5 -> 5 -> 5 -> 5 -> 5 -> 12.7 -> 12.7
>>> myList.removeDuplicates()
>>> myList
Head:Node(-9)
Tail:Node(12.7)
List:-9 -> -3.5 -> -3 -> 0 -> 1 -> 2 -> 3 -> 4 -> 4.5 -> 5 -> 12.7
>>> x.add(13)
>>> x.add(13)
>>> x
Head:Node(-9)
Tail:Node(13)
List:-9 -> -3.5 -> -3 -> 0 -> 0 -> 1 -> 2 -> 3 -> 3 -> 4 -> 4.5 -> 5 -> 12.7 -> 13 -> 13
>>> x.removeDuplicates()
>>> x
Head:Node(-9)
Tail:Node(13)
List:-9 -> -3.5 -> -3 -> 0 -> 1 -> 2 -> 3 -> 4 -> 4.5 -> 5 -> 12.7 -> 13
```