Project 2: Recursion

Due: Thursday, January 31st, 8:00 pm
This is not a team project, do not copy someone else's work.

Description

For this project, you will be implementing recursive functions for an ordered, singly linked list. Unlike in project 1, there is no LinkedList class. For each function, it takes in the head node of a linked list and uses recursion to perform an action. (To put in context, you used iterative algorithms in project 1.) Let it be clear - all of the functions you are implementing are **RECURSIVE**, so they call themselves. We have provided to you the LinkedNode class.

Turning It In

Your completed project must be submitted as a folder named "**Project2**" and must include:

- Recursion.py, a python3 file.
- README.txt, a text file that includes:
  - Your name and feedback on the project
  - How long it took to complete
  - A list of any external resources that you used, especially websites (make sure to include the URLs) and which function(s) you used this information for.

Assignment Specifications

The **LinkedNode class** is included in the skeleton file with the following functions and **should not be edited.**

- __init__(self, value, next_node=None)
  - This function initializes a node with a given value and next_node reference.
- __str__(self)
  - A node is represented in string form as 'value'. Use str(node) to make into a string.

You must complete and implement the following functions in **Recursion.py**. Take note of the specified return values and input parameters. **Do not change the function signatures.**
In addition to the Mimir testing, you will also be graded on the **run time and space** performance of your functions. See below what is expected for each function.

- **insert**(value, node=None)
  - Insert the given ***value*** into the linked list that has head ***node***
  - The value should be inserted at the end of the list
  - Return the starting node of the linked list
  - Time complexity O(n), Space complexity O(1)
- **string**(node)
  - Generate and return a string representation of the list, starting at ***node***
  - The values should be separated by a comma and a single space, with no leading or trailing comma
  - Time complexity O(n), Space complexity O(n)
- **remove**(value, node)
  - Remove the first node in the list with the given ***value*** starting at head ***node***
  - Return the starting node of the linked list
  - Time complexity O(n), Space complexity O(1)
- **remove_all**(value, node)
  - Remove all nodes in the list with the given ***value*** starting at head ***node***
  - Return the starting node of the linked list
  - Time complexity O(n), Space complexity O(1)
- **search**(value, node)
  - Looks for ***value*** in list starting with head ***node***
  - Returns True if the value is in the list and False if it is not in the list
  - Time complexity O(n), Space complexity O(1)
- **length**(node)
  - Calculates and returns the length of the list starting with head ***node***
  - Time complexity O(n), Space complexity O(1)
- **sum_all**(node)
  - Calculates and returns the sum of the list starting with head ***node***
  - Time complexity O(n), Space complexity O(1)
- **count**(value, node)
  - Counts and returns how many times the given ***value*** occurs in the list starting at head ***node***
  - Time complexity O(n), Space complexity O(1)
- **palindrome**(node, length, compare):
  - Determines if the list with head ***node*** and length ***length*** is a palindrome
  - Evaluate each node as valid; ie don't ignore any characters
  - **Compare** is a list used to hold comparison nodes - we use a list because it can be changed in a recursive call
  - Returns True if the linked list is a palindrome, False if it is not
  - Time complexity O(n), Space complexity O(1)

Assignment Notes

- You are required to add and complete the docstrings for each function. Use Project1 as a guideline to help you document your code.
- All of your functions must be recursive. You will loose all points related to the function if it is not recursive.

- Do not use additional data structures, such as lists, or strings (outside of the string() function). You will loose all points relating to the function if you do.
- Grading Rubric
  - 55 pts : Mimir visible test cases (see tests for point breakdown)
  - 20 pts : Additional hidden test cases
  - 25 pts : Manual grading (Complexity)
    - (5) Docstrings
    - (2 each) 1 for time/1 for space, insert, string, length, search, count, sum_all, remove, remove_all
    - (4) Palindrome

Written by Brandon Field and Erika Lustig