CSC 212 – Data Structures Spring 2017

Programming Project 3
Due no later than midnight 04.12.17

Doubly Linked List

(max score = 30 points)

For this project, you are to design and implement a doubly linked list. The list should be implemented in a class called DLList and should be modeled after the classes Node and UnorderedList which provided an implementation for a singly-linked list.

A doubly-linked list is one where each node has two references: one to the next node, and another to the previous node (You would need to modify the Node class accordingly (1 point)). The DLList class **must** contain a reference to the head element (as is the case for the singly-linked list) as well as a reference tail, which references the last element in the list. One advantage of having the tail reference is that it makes it more efficient to append elements to the list (there's no need to traverse the list from the beginning).

You are to implement the list methods below (22 points):

- `__init__(self):`
  List constructor. 1 point.
- `def add(self, item):`
  The method accepts as a parameter an item (not a node), which it inserts as the head element in the list. 1 points.
- `def size(self):`
  Returns the size of the list. 1 points.
- `def contains(self, item):`
  Given an item, the method returns True if the item is in the list, False otherwise. 2 points.
- `def rand_init(self, list_size):`
  Fill the list with `list_size` number of randomly generated integers in the range 1-100 (inclusive). If the list is non-empty, delete all elements first. As an example, given that `dList` is a reference to a list, the call `dList.rand_init(5)` would put 5 randomly generated values in the list. The list size after the function call should be 5. 2 points.
- `def remove_all(self, item):`
  The method removes all occurrences of item in the list. If item is not in the list, the method does nothing. For instance, if a list `dList` has the elements 4 4 3 4 5 4 3 4, then the call dList.remove(4) would change the list to 3 5 3. 3 points.
- `def __str__(self):`
  Returns a string representation of the list (use the Python list format, e.g. [a, b, c]). 2 points.
- `def append(self, item):`
  Append an item to the end of the list (Your method should use the tail reference). 1 points.
- `def slice(self, start, end):`
  Given two values start and end (where start < end), return the sublist starting at position start and ending at position end-1 as a Python list. Position 0 is the head of the list. The method should check for invalid inputs, in which case it should return `None`. For example, if a list `dList` has the elements 1 2 3 4 5 6 7, then the call `dList.slice(1, 3)` should return [2, 3], while `dList.slice(4, 4)` should return [], and `dList.slice(3, 10)` should return `None` since end > list size. 3 points.
- `def swap(self, i, j):`
  Given i and j, swap the list values at those positions. If either position is not valid, do nothing. As an example, if a list `dList` has the elements 1 2 3 4 5 6 7, then the call `dList.swap(0, 3)` would rearrange the list to 4 2 3 1 5 6 7. 2 points.

- `def shuffle(self):`
  Shuffle the contents of the list. Hint: you can call swap repeatedly using randomly generated list positions. The function should perform at least as many swaps as there are elements in the list. 2 points.
- `def is_palindrome(self):`
  Returns True if the elements in the list form a palindrome, False otherwise. 2 points.

Additionally, write a test program that demonstrates that the functions work correctly (3 points). Make sure you test each function using valid and invalid inputs. For example, to test `remove_all`, the following tests would be necessary:

- Remove element from an empty list
- Remove element that occurs multiple times in the list
- Remove an element that's not in the list
- Remove an element that occurs as the first element in the list
- Remove an element that occurs as the last element in the list

The test program **should not be interactive.** You shouldn't prompt the user to input any values.

Each of the methods should begin with a comment block that includes a description of the parameters, what the method does, and what it returns (if anything). Additionally, you should include comments within methods explaining the code. No line in your program should exceed a width of 79 characters. Comments and readability are worth 4 points.

Submission Instructions
Submit the python files containing the modified Node class, DLList and your test program as a zip archive named `project3.zip` (If you've implemented everything in a single python file, there's no need to create a zip archive. Just name the file `project3.py`). In a comment block at the beginning of the test program, include your name and the state of the implementation. Upload to the shared Google folder no later than **midnight on Wednesday 04.12.17**.