CSE 274: Data Abstraction and Data Structures
Program #2 – Linked List Algorithms - 120 Points
Due Sep 20, by 11:59 pm

## Outcomes:

- Write a Java program that implements linked list algorithms

## General requirements:

- Javadoc is NOT required for this project.
- Follow good programming practices - your source code will be evaluated for this project.
    - Format your code so that it is readable using generally accepted guidelines for formatting source code.
    - Keep your methods compact and cohesive. If a method starts to become long, or the nesting becomes too deep, consider creating a helper method.
    - Reuse existing methods, when feasible.
    - Your primary objective is clarity; efficiency is a secondary concern. That is, use the clearest solution if it does not change the big O.

## Specific requirements:

Download the class named **LinkedAlgorithms.java** and add a JUnit test file called "**JUnitTester.java**" to your project. You are to complete the methods in the LinkedAlgorithms.java file (read the comments above each method). In the implementation, you must use the linked list techniques discussed in class/video; all your code must be contained in LinkedAlgorithms.java. Do not use any Java collection classes (e.g., *ArrayList*, *LinkedList*, etc.). The linked list treats indices like an array: index 0 represents the first element in the list, 1 the second element, and so on.

## Notes:

- Do not put package statements in your code.
- All of your code must reside within **LinkedAlgorithms.java**. This and a JUnit test file called **JUnitTester.java** is all that you will submit.
- Do not delete any methods, rename any methods, change public to private, or change parameter lists. If you are not sure how to implement a method, leave the method as a stub (header plus return statement).
- You may add private methods as you see fit.
- You may not add any additional data members to the class definition.
- The main() method (located at the bottom of the class) must compile with your code.

## Scoring:

**(15 pts)** Programming style and neatness.

**(15 pts)** Appropriate and thorough JUnit testing

**(90 pts)** Correctness of methods (see below)

## Scoring Breakdown:

1. **(9 pts; 3 pts each) Constructors**
   a. `public LinkedAlgorithms()`
   b. `public LinkedAlgorithms(String [] data)`
   c. `public LinkedAlgorithms(LinkedAlgorithms original)`

2. **(16 pts; 4 pts each) Utilities**
   a. `public String toArray()`
   b. `public String toString()`
   c. `public int size()`
   d. `public boolean equalsLinkedList(LinkedAlgorithms other)`

3. **(4 pts; 2 pts each) Search**
   a. `public boolean contains(String data)`
   b. `public int find(String data)`

4. **(9 pts; 3 pts each) Retrieval**
   a. `public String getFirst()`
   b. `public String getLast()`
   c. `public String getAt(int i)`

5. **(15 pts) Insertion**
   a. (9 pts; 3 pts each) By position
      i. `public void insertFirst(String data)`
      ii. `public void insertLast(String data)`
      iii. `public void insertAt(int i, String data)`
   b. (6 pts; 3 pts each) By data value
      i. `public void insertBefore(String newData, String existingData)`
      ii. `public void insertAfter(String newData, String existingData)`

6. **(15 pts) Removal**
   a. (9 pts; 3 pts each) By position
      i. `public String removeFirst()`
      ii. `public String removeLast()`
      iii. `public String removeAt(int i)`
   b. (6 pts; 3 pts each) By data value
      i. `public boolean removeFirstOccurrenceOf(String data)`
      ii. `public int removeAllOccurrencesOf(String data)`

7. **(6 pts; 3 pts each) List Manipulation**
   a. `public void reverse()`
   b. `public void toUpper()`

8. **(16 pts; 4 pts each) Properties**
   a. `public String getConcatenation()`
   b. `public String getAlphabeticallyLast()`
   c. `public int indexOfAlphabeticallyLast()`
   d. `public boolean anagrams(LinkedAlgorithms other)`

## Turn-in:

You will turn in a zip file containing the **LinkedAlgorithms.java** and **JUnitTester.java**