CSCI 2110 Data Structures and Algorithms
Laboratory No. 6
Week of October 23-27, 2023

Due: Sunday, October 29, 11.59 PM

Ordered Lists Data Structure

This lab is on Ordered Lists, specifically the two-finger walking algorithm and its variations.

You will need two files for this lab:
OrderedList.java (this is the OrderedList class that was discussed in the lectures).
RandomNames.txt (text file containing a list of 2000 random first names).

They are given alongside this lab document. Download them.

Marking Scheme
Each exercise carries 10 points.
Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.
Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input

Submission Requirements:

- No submission other than a single ZIP file will be accepted.
- You MUST SUBMIT .java files that are readable by the TA. If you submit files that are unreadable such as .class file, the lab will be marked 0.
- Please additionally comment out package specifiers.

Late Submission Penalty: The lab is due on Sunday at 11.59 PM. Late submissions up to 5 hours (4.59 AM on Monday) will be accepted without penalty. After that, there will be a 10% late penalty per day on the mark obtained. For example, if you submit the lab on Monday at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past two days (48 hours) after the grace submission time, that is, submission past 4.59 AM Wednesday will not be accepted.

Follow these steps for the lab.

Step 0: Study the OrderedList.java class and understand the methods.

Step 1: Develop a class called Merge.java with the following three static methods based on the two-finger walking algorithm discussed in the lectures.

Method 1: Input arguments are two ordered lists; the method creates and returns a new list that contains the merger of list1 and list2, using the two-finger walking algorithm.

For this, you would write a method with the following header:

```
public static <T extends Comparable<T>> OrderedList<T> merge (OrderedList<T> list1,  OrderedList<T> list2)
    {
        //your code here
    }
```

The pseudocode is given in the lecture notes. Study the pseudocode and convert it into the method.

Method 2: Input arguments are two ordered lists; the method creates and returns a new list that contains the common items in list1 and list2. You would still use the two-finger walking algorithm concept with a small variation in the pseudocode.

The header for this method would be:

```
public static <T extends Comparable<T>> OrderedList<T> common (OrderedList<T> list1,  OrderedList<T> list2)
    {
        //your code here
    }
```

Method 3: Input arguments are two ordered lists; the method creates and returns a new list that contains the items in list1 but not in list2. You would still use the two-finger walking algorithm concept with a small variation in the pseudocode.

The header for this method would be:

```
public static <T extends Comparable<T>> OrderedList<T> difference (OrderedList<T> list1, OrderedList<T> list2)
    {
        //your code here
    }
```

As an example, suppose we have the following lists:
list1: [Amar, Boris, Charlie, Dan, Fujian, Inder, Pei, Travis]
list2: [Alex, Betty, Charlie, Dan, Travis, Zola, Zulu]
then the first method (merge) should return the list
[Alex, Amar, Betty, Boris, Charlie, Dan, Fujian, Inder, Pei, Travis, Zola, Zulu]

The second method (common) should return the list
[Charlie, Dan, Travis]

The third method (difference) should return the list
[Amar, Boris, Fujian, Inder, Pei]

As you may notice from the above example, the items can be repeated across the two lists, but they are not repeated within each list. Same with the result lists – items are not repeated.

Step 2: Write a small main method, ask user input to create list1 and list2, and test the methods that you created to ensure that they work properly. You can use the above simple example lists and other boundary cases to test them. This is just for your own testing.

Step 3: You are given a text file RandomNames.txt containing a list of 2000 random names. This text file has been composed from a random name generator given in https://catonmat.net/tools/generate-random-names

Write a class called Merge.java that does the following:

a) Read the RandomNames.txt file and store the words in an ArrayList.

b) Create the first ordered list, list1: For this, prompt the user to enter an integer n1 between 1000 and 1500. Randomly select n1 words from the ArrayList containing the words and create an ordered list from this set of random names. Make sure that the words are not repeated in list1.

b) Repeat the same to create the second ordered list, list2: That is, prompt the user to enter an integer n1 between 1000 and 1500. Randomly select n2 words from the ArrayList containing the words and create an ordered list from this set of random names. Make sure that the words are not repeated in list2.

c) Merge list1 and list2 by calling Method 1. Save the resulting list in a text tile called merged.txt.
d) Find the common words in list1 and list2 by calling Method 2. Save the resulting list in a text file called common.txt.
e) Find the difference between list1 and list2 by calling Method 3. Save the resulting list in a text file called difference.txt.

What to submit:
One zip file containing the following:
1. OrderedList.java with the three methods, Method1, Method2 and Method3 added.
2. Merge.java
3. merged.txt
4. common.txt
5. difference.txt

You MUST SUBMIT .java files that are executable by your TAs. TAs will run your program on the RandomNames.txt file and check your solution. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers.