Data Structures and Algorithms, Spring '23 section1 Dr. Hristescu
Lab 8

Assigned: Thursday, March 23
Due: 10pm on Monday, March 27

General Notes:

- Solutions to different problems than specified in the description of this lab will not receive any credit. Follow the instructions carefully.
- For Problems 1 and 2 have the search handled by the driver. Therefore, the functionality of the List ADT should not be changed for Problems 1 and 2. For all problems use **String** as your item type.
- Don't start from scratch, use the frame of the Driver from Lab 2, but note that there is no option to reverse the list. Only the highlighted menu options need to be addressed. All other options have already been implemented for Lab 2.
- For ALL problems design a **search** method (in the driver for P1 and P2 and in the list implementation for P3) to handle the search. For all problems (P1, P2, and P3) the search method should return information about the outcome of the search as well as positional information for a successful search (in all cases) as well as unsuccessful search in an ordered collection (P2,3). The information should be **encoded in ONE integer value that is returned**. The value returned by the search method should be decoded for successful/unsuccessful outcome and positional information **without additional probing** of the collection. Clearly document the search method with the returned value. The documentation of the returned value should be descriptive enough to be used in designing the application program and know how to decode it **without having to probe the collection**.
  For P1 and P2 use the following signature for search:
  public static int search(String key, ListArrayBasedPlus list)
- Since you are working with String type data, you should use compareTo for item comparisons.Note that equals can be used for equality comparison as well, but compareTo covers both equality and inequality comparisons.

<u>Problem1:</u> Use the **ListRA** code developed for **Lab 2** as is (without any changes). Enhance the application program from Lab 2 with an option to search an **unordered** list of unique items for a user specified item. Use **String** as your item type. Therefore, the menu options should be:

0. Exit program
1. Insert item into the list
2. Remove item from the list
3. Get item from the list
4. **Search for a specific item in the list**
5. Clear the list
6. Print size and content of the list

For **menu option 4**, the user has to specify an item to search for in the list. If the item is in the list, then the driver should report the position in the list where the item was found. If the item is not in the list, the user should be notified of that fact. Call the **search** method for option 4 and display the required information based on the **returned** int value without additional probing of the collection.

<u>Problem2:</u> Modify the program from **Problem 1** to work with an **ordered** list of unique items (use ***ascending***/*increasing* order). Therefore, for **menu option 1**, the user should **only specify the item that has to be inserted**. For option 1 the application program **should call the search method** to find, if the item is not already in the list, the correct position in the list where the item has to be inserted in order to maintain the ascending order in the list and then insert the item. If the item is already in the list, **duplicates should NOT be inserted.** The Driver should use the info in the integer returned by the search method without additional probing of the collection.
The menu options should be:

0. Exit program
1. **Insert item into ordered list**
2. Remove item from the list
3. Get item from the list
4. **Search for a specific item in the list**
5. Clear the list
6. Print size and content of the list

Implement either A or B. Clearly document your choice. **Do NOT use binary search for this problem.**
A: Use Modified Sequential Search I or II discussed and analyzed in class (eagerly checking for a stopping condition- equality for successful or smaller current key for unsuccessful)
B: Use Modified Sequential Search III discussed and analyzed in class (eagerly advancing). [recommended for efficiency]

<u>Problem 3:</u> Design an **AscendinglyOrderedStringList ADT** of **unique items** and implement it using an array-based implementation. The functionality of this ADT should be enhanced with a **search** method that finds (using **binary**

search):
- If the item is in the list: the position in the list where the item is located
- If the item is not in the list: the position where the item can be inserted in order to maintain the ascending order of the items in the list.

In your AscendinglyOrderedStringList ADT implementation, the **add** method **has only one parameter**. The add method should use the **search** method to find, if the item is not already in the list, the position in the list where the specified item has to be inserted and inserts it there. If the item is already in the list, it **should NOT insert a duplicate**. Use the info returned by the search method without additional probing of the collection. Use this new AscendinglyOrderedStringList interface:

```
public interface AscendinglyOrderedStringListInterface
{ public boolean isEmpty();
  public int size();
  public void add(String item) throws ListIndexOutOfBoundsException;
  public String get(int index) throws ListIndexOutOfBoundsException;
  public Object remove(int index) throws ListIndexOutOfBoundsException;
  public int search(String key);
  public void removeAll();
}
```

Test the functionality of your class using a menu-driven test application with the options below. Adapt your application program from Problem 1 to work with an ordered list of **String**s (assume lexicographically ascending order).

0. Exit program
1. **Insert specified item into the list**
2. Remove item in specified position in the list
3. **Search list for a specified item**
4. Clear the list
5. Display the content of the list

Note:
- For insert (menu option 1), the user should only specify the item that has to be inserted.
- Build on the array-based implementation of the List ADT from Lab 2.

Implement either A or B. Clearly document your choice.
A: Use **iterative** Binary Search I presented and discussed in class (eager equality checking).
B: Use **iterative** Binary Search II presented and discussed in class (eager advancing and postponing equality checking until the sublist is of size 1).[recommended for efficiency] *make sure to test that the search returned the correct positional information if the key is a) smaller than the last item but larger than the next to last item and b) larger than the last item of the collection

Problem4: **Javadoc-Comment** your **AscendinglyOrderedStringList** class and generate Java Documentation using Javadoc on elvis (instructions). Pay special attention to the **search** method – clients looking at the java doc should be able to know how to decode the returned integer value **without additional probing of the list**. Use the guidelines in http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

Problem5: **Lab8Conclusions**: Summarize in a few sentences what you have learned from working on this lab.

Extra Credit: (marked clearly ECI and ECII and noted in file header)
ECI: Make your no-duplicates AscendinglyOrderedStringList into a generic class. Call it AscendinglyOrderedList. Use an **Item** class structured like the textbook's Person class (page 596) that is a subclass of the abstract class **KeyedItem**. Use as the **Comparable** key a **String** (passed to the super constructor as a parameter) just like in Person and add two more [otherwise useless fields for our search problem] data fields one of type **boolean** named **assocboolean** and the other one of type **int** named **assocint** but with appropriate getters and setters associated with them as well as a toString method that returns the information in the format **{key/assocboolean/assocint}.**
Use the declaration below and **AscendinglyOrderedList<Item,String>** in your Driver.

```
public class AscendinglyOrderedList<T extends KeyedItem<KT> , KT extends
Comparable <? super KT>> implements AscendinglyOrderedListInterface<T, KT>
```

Use:

```
public interface AscendinglyOrderedListInterface<T,KT>
{ boolean isEmpty();
  int size();
  void add(T item) throws ListIndexOutOfBoundsException;
  T get(int index) throws ListIndexOutOfBoundsException;
  void remove(int index) throws ListIndexOutOfBoundsException;
  int search(KT key);
  void clear();
}
```

ECII: Redesign the **AscendinglyOrderedStringList** ADT to **allow duplicates**. Call it **AscendinglyOrderedStringListD**.

Submit: Follow the instructions to turn in the lab electronically using **Lab8** for the name of your directory.

Develop your own extensive input suite and test your solutions thoroughly on it. Submit all the code developed for this lab and all sample runs on the input suite you developed. Make sure you include code and testing for **AscendinglyOrderedStringList,** and if EC is included also **AscendinglyOrderedList** and **AscendinglyOrderedStringListD,**