Implementation of data structures and algorithms
Spring 2023
Short Project 2: Linked lists

Version 1.0: Initial description.

Due: 11:59 PM Friday, Feb 3, 2023.

Submission procedure:

* Create a folder whose name is your netid (NId).
* Place all files you are submitting in that folder.
* Use "package NId;" in all your java files.
* There is no need to submit binary files created by your IDE (such as class files).
* Include a text file named "readme.txt", that explains how to compile and run the code.
* Zip the contents into a single zip or rar file.
* If the zip file is bigger than 1 MB, you have included unnecessary files.
* Delete them and create the zip file again.
* Upload the zip or rar file on elearning.
* Submission can be revised before the deadline.
* The final submission before the deadline will be graded.
* Only one member of each team needs to submit project.
* Include the names of all team members in ALL files.
* Write quality code. Follow the Javadoc standards.

Team task:
1. Extend the SinglyLinkedList.java (in shared folder) to a doubly linked list (DoublyLinkedList class). Use class inheritance to avoid unnecessary duplication of code. Write a method dllIterator( ), which extends the operations done by SLLIterator and adds methods hasPrev(), prev(), and add(x). The add method of the iterator should insert x before the element that will be returned by a call to next().

Additional Practice problems (no need to submit):

2. Given two linked lists implementing sorted sets, write functions for union, intersection, and set difference of the sets.

```
public static<T extends Comparable<? super T>>
   void intersect(List<T> l1, List<T> l2, List<T> outList) {
         // Return elements common to l1 and l2, in sorted order.
         // outList is an empty list created by the calling
         // program and passed as a parameter.
         // Function should be efficient whether the List is
         // implemented using ArrayList or LinkedList.
         // Do not use HashSet/Map or TreeSet/Map or other complex data structures.
       }

public static<T extends Comparable<? super T>>
```

```
    void union(List<T> l1, List<T> l2, List<T> outList) {
          // Return the union of l1 and l2, in sorted order.
          // Output is a set, so it should have no duplicates.
        }

  public static<T extends Comparable<? super T>>
    void difference(List<T> l1, List<T> l2, List<T> outList) {
          // Return l1 - l2 (i.e, items in l1 that are not in l2), in sorted order.
          // Output is a set, so it should have no duplicates.
        }
```

3. Write the Merge sort algorithm that works on linked lists. This will be a member function of a linked list class, so that it can work with the internal details of the class. The function should use only O(log n) extra space (mainly for recursion), and not make copies of elements unnecessarily. You can start from the SinglyLinkedList class provided or create your own.

```
  static<T extends Comparable<? super T>> void mergeSort(SortableList<T> list) { ... }
```

Here is a skeleton of SortableList.java:

```
  public class SortableList<T extends Comparable<? super T>> extends SinglyLinkedList<T> {
  void merge(SortableList<T> otherList) {  // Merge this list with other list
  }
  void mergeSort() { Sort this list
  }
  public static<T extends Comparable<? super T>> void mergeSort(SortableList<T> list) {
        list.mergeSort();
  }
}
```

4. Extend the "unzip" method to "multiUnzip" method in the SinglyLinkedList class:

```
  void multiUnzip(int k) {
        // Rearrange elements of a singly linked list by chaining
        // together elements that are k apart.  k=2 is the unzip
        // that is given.  If the list has elements
        // 1..10 in order, after multiUnzip(3), the elements will be
        // rearranged as: 1 4 7 10 2 5 8 3 6 9.  Instead if we call
        // multiUnzip(4), the list 1..10 will become 1 5 9 2 6 10 3 7 4 8.
  }
```

5. Write recursive functions for the following tasks:
   (i) reverse the order of elements of the SinglyLinkedList class,
   (ii) print the elements of the SinglyLinkedList class, in reverse order.
   Write the code and annotate it with proper loop invariants.
   Running time: O(n).