COSC 237 – ASSIGNMENT 03

Submission Instruction

Failure to follow the below instructions will result in a 20% penalty.

1. Do each programming problem in a single file with the name format: ***"<your_last_name>_HW_XX_YY.java"*** where ***XX*** is the assignment number and ***YY*** is the problem number. For instance, for assignment 01 and student name "David Smith", the filenames would be "Smith_HW_01_01.java", "Smith_HW_01_02.java", etc for questions #1, #2, and so on.
2. When you are done, create a folder named ***"<first_name>_<last_name>_HW_XX"*** and put all your ***.java*** and input files or test files (NO ***.o*** or ***.class*** files) in this folder. For example: folder name *"David_Smith_HW_01"* for student name "David Smith". No *.class* files or test files should be included.
3. Finally, create a ZIP file of this folder with the same name (i.e., "David_Smith_HW_01.zip") and submit it over Blackboard. If you have an issue, change the file extension to ***.txt*** (i.e., change "David_Smith_HW_01.zip" to "David_Smith_HW_01.txt") and resubmit.

For this assignment, you will work with a partner and both of you should contribute significantly to the solution for every question. I suggest that both of you do the programs alone, pass your version to your partner, criticize the version written by your partner, and present a solution that has the best features. In principle, you should not discuss the homework with anyone, except your homework partner and your instructor. If you do discuss it with someone else, you should say this in the preamble of the homework. You should only submit work that is completely your own and you should be able to explain all your homework to me. Make sure you handle **invalid input, including type mismatch** - input validation is a requirement! Focus on good design, good style, and, most importantly, reuse of code. Start early and remember the rules about late assignments/syllabus → **no late assignments!**

Question 1 [40pts] Modify the class **`UnorderedLinkedListInt`** in the lecture (file "Lecture_Array_Based_Lists.pdf") by adding 2 more methods to concatenate/merge 2 unsorted linked lists and one to split/divide a list into 2 lists according to a key.

- Method signature: `public void merge1(UnorderedLinkedListInt list)`. This method will append `list` to the original list. Side effect: the original list is lost.
- Method signature: `public UnorderedLinkedListInt merge2(UnorderedLinkedListInt list)`. This method should create a new list with the concatenation result.
- Method signature: `public void split(UnorderedLinkedListInt list1, UnorderedLinkedListInt list2, int key)`. This method should split the original list into `list1` and `list2`. Follow the same rules as in the previous problems.

Use this client to test the second method to merge:

```
import java.util.*;
public class ClientMerge_2 {
    static Scanner input = new Scanner(System.in);
    public static void main(String[] args) {
        UnorderedLinkedListInt list1 = new UnorderedLinkedListInt();
        UnorderedLinkedListInt list2 = new UnorderedLinkedListInt();
        UnorderedLinkedListInt list3 = new UnorderedLinkedListInt();
        int num;
        System.out.println("Enter integers for the first list(999 to stop)");
        num = input.nextInt();
        while (num != 999) {
            list1.insertLast((Integer) num);
            num = input.nextInt();
        }
        System.out.println("Enter integers for the second list(999 to stop)");
        num = input.nextInt();
        while (num != 999) {
            list2.insertLast((Integer) num);
            num = input.nextInt();
        }
        System.out.print("\nThe first list is: ");
        list1.print();
        System.out.println("\nThe length of the first list is: " + list1.length());
        if (!list1.isEmptyList()) {
            System.out.println("First element/list1: " + list1.front());
            System.out.println("Last element/list1: "  + list1.back());
```

```
        }
        System.out.print("\nThe second list is: ");
        list2.print();
        System.out.println("\nThe length of the second list is: " + list2.length());
        if (!list2.isEmptyList()) {
            System.out.println("First element/list2: " + list2.front());
            System.out.println("Last element/list2: "  + list2.back());
        }
        list3 = list1.merge2(list2);
        System.out.print("\nAfter concatenating the 2 lists, the merged list1 is: ");
        list3.print();
        System.out.println("\nThe length of the merged list is: " + list3.length());
        if (!list3.isEmptyList()) {
            System.out.println("First element/merged list: " + list3.front());
            System.out.println("Last element/merged list: "  + list3.back());
        }
        System.out.println("Enter key for split: ");
        num = input.nextInt();
        list3.split(list1, list2, num);
        System.out.print("\nThe first list after split is: ");
        list1.print();
        System.out.print("\nThe second list after split is: ");
        list2.print();
        System.out.println();
    }
}
```

SAMPLE OUTPUT:

```
Enter integers for the first list(999 to stop)
37 10 88 59 27 20 14 32 89 100 12 999
Enter integers for the second list(999 to stop)
23 56 34 15 78 19 999

The first list is: 37 10 88 59 27 20 14 32 89 100 12
The length of the first list is: 11
First element/list1: 37
Last element/list1: 12

The second list is: 23 56 34 15 78 19
The length of the second list is: 6
First element/list2: 23
Last element/list2: 19

After concatenating the 2 lists, the merged list1 is: 37 10 88 59 27 20 14 32 89 100 12 23
56 34 15 78 19
The length of the merged list is: 17
First element/merged list: 37
Last element/merged list: 19
```

```
Enter key for split: 20
The first list after split is: 10 20 14 12 15 19
The second list after split is: 37 88 59 27 32 89 100 23 56 34 78
```

Write a slightly modified version of this client to test the first method to merge.

Question 2 [60 pts] Doubly linked list programming assignment: Rewrite the implementation of a sorted list using a doubly linked list. Use as a guide the implementation with a single linked list (lecture notes).

Definition: A doubly linked list is a list in which each node is linked to both its successor and its predecessor.

Example:

In this definition, a node has 2 link members; one is a reference to the next node in the list--immediate successor, and the other is a reference to the previous node in the list--immediate predecessor. The first node in the list has no predecessor and the last node has no successor. The advantage of using a double linked list is that the list can be traversed from first node to last, as any linked list, AND ALSO from last node to first (very convenient when we need to traverse a list backwards). The disadvantage is the extra storage required for the second reference in each node. Also, the inserting and deleting operations are a bit slower (both references have to be updated).

Operations to be implemented on a double linked list:

1. Initialize the list
2. Determine whether the list is empty
3. Retrieve the first element in the list
4. Retrieve the last element in the list
5. Find the length of the list
6. Display the list (direct and reverse order)
7. Search the list for a given item
8. Insert an item in the list
9. Delete an item from the list
10. toString (direct and reverse order, 2 versions for each, one using recursion)
11. Compare 2 lists for equality (equals)
12. Make a copy of the list
13. Make a copy of the list in reverse order

Use this interface:

```
public interface DoubleLinkedListADT<T>  {
    public void initializeList();
    public boolean isEmptyList();
    public T front();
    public T back();
    public int length();
    public void print();
```

```
    public void reversePrint();
    public boolean search(T searchItem);
    public void insertNode(T insertItem);
    public void deleteNode(T deleteItem);
    public String toString();
    public String recursiveToString();
    public String backwardsString();
    public String recursiveBackwardsString();
    public boolean equals(Object o);
    public void copy(DoubleLinkedList<T> otherList);
    public void reversedCopy(DoubleLinkedList<T> otherList);
}
```

Complete this class definition:

```
public class DoubleLinkedList<T> implements DoubleLinkedListADT<T> {
    //Double linked list node class
    public class DoubleLinkedListNode<T>  {
        T info;
        DoubleLinkedListNode<T> next;
        DoubleLinkedListNode<T> back;

        public DoubleLinkedListNode() {
            info = null;
            next = null;
            back = null;
        }

        public String toString() {
            return info.toString();
        }
    }

    protected int count;  //number of nodes
    protected DoubleLinkedListNode<T> first; //reference to first node
    protected DoubleLinkedListNode<T> last;  //reference to last node
    ...
    ...
    ...
}
```

Use this client program to do the testing:

```
public class Client_DLLString {
    public static void main(String[] args) {
       DoubleLinkedList<String>list_1 = new DoubleLinkedList<String>();
       DoubleLinkedList<String>list_2 = new DoubleLinkedList<String>();
       String item;
       list_1.insertNode("John");
       list_1.insertNode("Ann");
       list_1.insertNode("Paul");
```

```
        list_1.insertNode("Joshua");
        list_1.insertNode("Will");
        list_1.insertNode("Emma");
        list_1.insertNode("Peter");
        list_1.insertNode("Linda");
        list_1.insertNode("Joan");
        list_1.insertNode("David");
        list_1.insertNode("Miriam");
        list_1.insertNode("Leah");
        list_1.insertNode("Jane");
        System.out.println("Inserted in first list the names: John, Ann, Paul, Joshua,
Will, Emma, Peter, Linda, Joan, David, MIriam, Leah, Jane");
        System.out.println("(Testing toString) First list sorted is: " + list_1);
        System.out.println("(Testing recursive toString) First list sorted is: [head] - " +
list_1.recursiveToString());
        System.out.println("(Testing backwards) First list reversed (print) is: " +
list_1.backwardsString());
        System.out.print("(Testing recursive backwards) First list reversed (print) is: " +
list_1.recursiveBackwardsString());
        System.out.println(" - [head]");
        System.out.println("Will copy in second list only names that don't start with J.
List one destroyed.");
        while (!list_1.isEmptyList()) {
            item = list_1.front();
            list_1.deleteNode(item);
            if(item.charAt(0) != 'J')
                list_2.insertNode(item);
        }
        System.out.println("Second list should hold names that don't start with J (sorted):
" + list_2);
        System.out.println("First list should be empty. Nothing printed. ");
        list_1.print();
        System.out.print("(Testing equals) ");
        if(list_1.equals(list_2))
            System.out.println("The 2 lists are equals");
        else
            System.out.println("The 2 lists are NOT equals");
        System.out.print("(Testing copy) ");
        list_1.copy(list_2);
        System.out.println("First list after copy is: " + list_1);
        System.out.print("(Testing reversed copy) ");
        list_1.reversedCopy(list_2);
        System.out.println("First list as the copy of the second list reversed is: " +
list_1);
    }
}
```

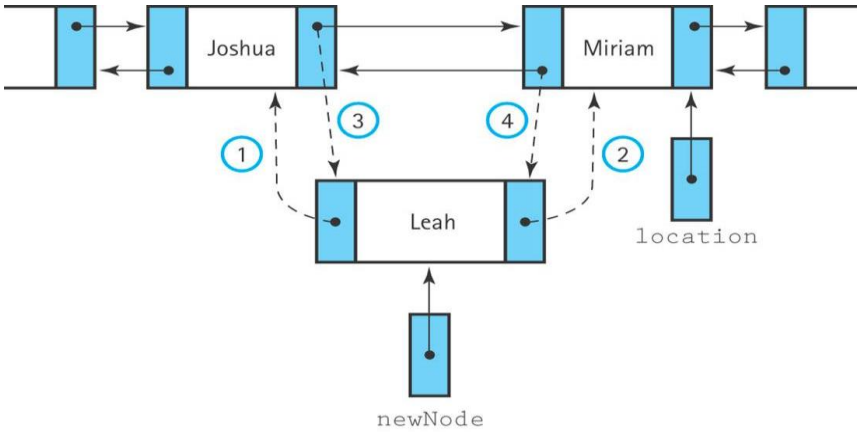
OUTPUT:

```
Inserted in first list the names: John, Ann, Paul, Joshua, Will, Emma, Peter, Linda, Joan,
David, MIriam, Leah, Jane
```

```
(Testing toString) First list sorted is: [head] - Ann - David - Emma - Jane - Joan - John
- Joshua - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
(Testing recursive toString) First list sorted is: [head] - Ann - David - Emma - Jane -
Joan - John - Joshua - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
(Testing backwards) First list reversed (print) is: [tail] - Will - Peter - Paul - Miriam
- Linda - Leah - Joshua - John - Joan - Jane - Emma - David - Ann - [head]
(Testing recursive backwards) First list reversed (print) is: [tail] - Will - Peter - Paul
- Miriam - Linda - Leah - Joshua - John - Joan - Jane - Emma - David - Ann - [head]
Will copy in second list only names that don't start with J. List one destroyed.
Second list should hold names that don't start with J (sorted): [head] - Ann - David -
Emma - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
First list should be empty. Nothing printed.
(Testing equals) The 2 lists are NOT equals
(Testing copy) First list after copy is: [head] - Ann - David - Emma - Leah - Linda -
Miriam - Paul - Peter - Will - [tail]
(Testing reversed copy) First list as the copy of the second list reversed is: [head] -
Will - Peter - Paul - Miriam - Linda - Leah - Emma - David - Ann - [tail]
```

Hints:

Take a look at the following picture when you write the method to insert a node:



Take a look at the following picture when you write the method to delete a node: