Problem 1 (0.75 points)

Write the code for the public static void reverseWord(char[] word, int first, int last) method. This method reverse the word contained in the char array word. Notice that you do not have to return a new array, but reverse the given one. You can assume that first and last are always positive numbers or zero, and that last will be smaller than the array length.

You MUST implement this method using recursion. Solutions using iterative or any other approaches will not be graded.

You do not need to create new arrays. Solutions based on creating new arrays will be penalized.

You can see an example of use below:
```java
public class TestingRecursion {

    public static void main(String[] args) {
        char[] word = {'f', 'l', 'o', 'w', 'e', 'r'};
        reverseWord(word, 0, word.length - 1);
        System.out.println(new String(word)); // Prints rewofl

        char[] word2 = {'w', 'h', 'e', 'e', 'l'};
        reverseWord(word2, 0, word2.length - 1);
        System.out.println(new String(word2)); // Prints leehw

        char[] word3 = {'e', 'l', 'e', 'v', 'a', 't', 'o', 'r'};
        reverseWord(word3, 1, 4);
        System.out.println(new String(word3)); // Prints eaveltor
    }

    public static void reverseWord(char[] word, int first, int last) {
    }
}
```
Problem 2 (2 points)

Observe carefully the `Node<E>` and `ExamLinkedList<E>` classes. Implement the public `double calculateElementsAverage(E searchedContent)` method in `ExamLinkedList<E>` class. This method must average the `value` attribute of every node whose `content` is equal to the one passed as parameter, `searchedContent`. This is, the method must find the nodes which content is equal to `searchedContent`, and calculate the average of the `values` stored in such nodes. The `content` of each node can be anything, including `null`.

Solutions that define and/or use new attributes or new methods (except for `equals()`) will be penalized.

```java
public class Node<E> {
    public E content;
    public int value;
    public Node<E> following;

    public Node(E info, int value, Node<E> next) {
        this.content = info;
        this.value = value;
        this.following = next;
    }
}
```
public class ExamLinkedList<E> {
    private Node<E> first;

    public ExamLinkedList() { this.first = null; }

    public double calculateElementsAverage(E searchedContent) {

    }
}
Problem 3 (2.25 points)

Observe carefully the code for the \texttt{BSTNode\langle E \rangle} (Binary Search Tree Node) and \texttt{BSTree\langle E \rangle} (Binary Search Tree) classes below. Assume that the root of every subtree will always have two child subtrees, which can be empty or not, but cannot be null.

```java
public class BSTNode\langle E \rangle {
    private E info;
    private String key;
    private BSTree\langle E \rangle left;
    private BSTree\langle E \rangle right;

    public BSTNode(E info, String key) {
        this.info = info;
        this.key = key;
        this.left = new BSTree\langle E \rangle();
        this.right = new BSTree\langle E \rangle();
    }

    public E getInfo() { return this.info; }
    public String getKey() { return this.key; }
    public BSTree\langle E \rangle getLeft() { return this.left; }
    public BSTree\langle E \rangle getRight() { return this.right; }

    public void setLeft(BSTree\langle E \rangle left) {
        if(left == null) {
            this.left = new BSTree\langle E \rangle();
        } else {
            this.left = left;
        }
    }

    public void setRight(BSTree\langle E \rangle right) {
        if(right == null) {
            this.right = new BSTree\langle E \rangle();
        } else {
            this.right = right;
        }
    }
}

public class BSTree\langle E \rangle {
    private BSTNode\langle E \rangle root;

    public BSTree() { this.root = null; }
    public BSTree(E info, String key) {
        this.root = new BSTNode\langle E \rangle(info, key);
    }
}
```
public boolean isEmpty() { return (this.root == null); }

public String toStringAlphabetical(boolean reverse) {
    /* Your code here */
}

Section 1 (0.1 points)

Which kind of tree traversal would you use in order to obtain the nodes of a tree in alphabetical order of its keys?

Section 2 (2.15 points)

Write the code for the public String toStringAlphabetical(boolean reverse) method. This method will return a string with the textual representation of the tree in alphabetical order of its keys, with the format key1:info1 key2:info2 key3:info3 ... (notice that there is only one space between elements). The parameter reverse indicates if the alphabetical order is reverse (true) or not (false). If the tree is empty, the method must return an empty string.

For instance, the string returned by this method for the tree

would be

BSTree<String> examTree = new BSTree<String>();
/** Tree construction */
...

String strAlphabetical = examTree.toStringAlphabetical(false));
// strAlphabetical would be B:d D:b E:e F:a P:c R:f

String strAlphabeticalReverse = examTree.toStringAlphabetical(true));
// strAlphabeticalReverse would be R:f P:c F:a E:e D:b B:d

public String toStringAlphabetical(boolean reverse) {

}
**Answer to problem 1**

```java
public class TestingRecursion {

    public static void main(String[] args) {
        char[] word = {'f', 'l', 'o', 'w', 'e', 'r'};
        reverseWord(word, 0, word.length - 1);
        System.out.println(new String(word)); // Prints rewolf

        char[] word2 = {'w', 'h', 'e', 'e', 'l'};
        reverseWord(word2, 0, word2.length - 1);
        System.out.println(new String(word2)); // Prints leehw

        char[] word3 = {'e', 'l', 'e', 'v', 'a', 't', 'o', 'r'};
        reverseWord(word3, 1, 4);
        System.out.println(new String(word3)); // Prints eaveltor
    }

    public static void reverseWord(char[] word, int first, int last) {
        if (first >= last) {
            // Stop condition. Might be this one or any equivalent return;
        } else {
            char aux = word[first];
            word[first] = word[last];
            word[last] = aux;
            reverseWord(word, first + 1, last - 1);
        }
    }
}
```

**Evaluation criteria for problem 1**

The problem is graded from 0 to 7.5, being 0 the lowest mark and 7.5 the highest.

The student mark starts at 0, then the following modifiers are applied in order:

[C100] (+2) Stop condition defined and correctly applied.

[C102] (+1) Define auxiliary char for storing first or last char.

[C103] (+1) Swipe (current) first and last letters.

[C104] (+2.5) Recursive call: 1 call to the recursive method, 0.5 for using same array, 0.5 for using first+1, 0.5 for using last-1.

[C105] (+1) Not returning anything.

[C106] (-1) (may be applied several times) For each time the naming conventions are not applied, or for code style and clarity missing.

[C107] (-1) If new arrays are created.
(grade=0) If iterative or any no recursive approach is used.

Answer to problem 2

```java
public class ExamLinkedList<E> {
    private Node<E> first;

    public ExamLinkedList() { this.first = null; }

    public double calculateElementsAverage(E searchedContent) {
        Node<E> current = this.first;
        double average = 0;
        double sum = 0;
        double count = 0;
        while (current != null) {
            if (searchedContent != null) {
                if (searchedContent.equals(current.content)) {
                    count++;
                    sum += current.value;
                }
            } else {
                if (current.content == null) {
                    count++;
                    sum += current.value;
                }
                current = current.following;
            }
            if(count>0) {
                average = sum/count;
            }
        }
        return average;
    }
}
```

Evaluation criteria for problem 2

The problem is graded from 0 to 20, being 0 the lowest mark and 20 the highest.

The student mark starts at 0, then the following modifiers are applied in order:

[C200] (+1) Define reference to current node.

[C201] (+1) Define reference to current node as Node<E> (use of generics).

[C202] (+2) Define count and sum variables outside the list traversal (1) as doubles (1).
Answer to problem 3

Section 1

Inorder

Section 2

```java
class BSTree<E> {
    private BSTNode<E> root;
	public BSTree(){ this.root = null; }
	public BSTree(E info, String key) {
		this.root = new BSTNode<E>(info, key);
	}
	public boolean isEmpty() { return (this.root == null); }
	public String toStringAlphabetical(boolean reverse) {
		String leftSubtree = "";
```
String current = "";
String rightSubtree = "";

if (!this.isEmpty()) {
    current = this.root.getKey() + ":" + this.root.getInfo();

    leftSubtree = this.root.getLeft().toStringAlphabetical(reverse);
    if (!leftSubtree.equals("")) {
        leftSubtree = leftSubtree + " ";
    }

    rightSubtree = this.root.getRight().toStringAlphabetical(reverse);
    if (!rightSubtree.equals("")) {
        rightSubtree = rightSubtree + " ";
    }
} else {
    return "";
}

if (reverse) {
    return rightSubtree + current + leftSubtree;
} else {
    return leftSubtree + current + rightSubtree;
}

---

**Evaluation criteria for problem 3**

The problem is graded from 0 to 21.5, being 0 the lowest mark and 21.5 the highest.

The student mark starts at 0, then the following modifiers are applied in order:

[C300] (+2.5) Stop condition defined and correctly applied (empty tree).

[C301] (+2.5) Obtain current node’s textual representation: 0.5 accessing root, 0.5 accessing key, 0.5 applying ":" format, 0.5 accessing root, 0.5 accessing content.

[C302] (+2.5) Obtain left subtree’s textual representation: 0.5 accessing root, 0.5 accessing left subtree, 1 applying recursive method, 0.5 passing reverse as parameter.

[C303] (+2.5) Obtain right subtree’s textual representation: 0.5 accessing root, 0.5 accessing right subtree, 1 applying recursive method, 0.5 passing reverse as parameter.

[C304] (+1) Add space after left subtree representation, if left subtree is not empty.
[C305] (+1) Add space after right subtree representation, if right subtree is not empty.

[C306] (+2) Return empty string if tree is empty.

[C307] (+1) Check if order required is reverse or not.

[C308] (+1) Return clauses (for both orders).

[C309] (+1.5) Reverse inorder (right - current - left).

[C310] (+1.5) Straight inorder (left - current - right).

[C311] (+2.5) Format applied (spaces between right/left - current - left/right).

[C312] (-1) (may be applied several times) For each time naming conventions are not applied, or for code style and clarity missing.