Alec DiPasquale

Programming Project #1

Overall description of the project	8
Test plan (list of test cases)	0
Source code that correctly implements the four operations: isEmpty(), howMany(), uniqInsert() and smallest() - including comments	67
Source code of testing program	9
Total	84

Good work on coding the new operations – see additional feedback below.

For this assignment, a brief description of the StringLog ADT should be part of the project description.

Test plan / list of test cases not submitted.

Testing program did not test uniqInsert() return value, nor case insensitivity of howMany() and uniqInsert().

You may submit updated documentation (revised description and list of test cases) to improve grade.

```
//new isEmpty() method by Alec

//PRECONDITIONS: none
public boolean isEmpty()
//Returns true if Linked List is empty, false otherwise
{
  LLStringNode node;
  node = log;
  if(node == null)
    return true;
  else
    return false;
}
```

Simply testing log equal to null works, and is a bit simpler.

Feedback continues on next page.

```
  //new uniqInsert(String element) method by Alec
  //PRECONDITIONS: none
public boolean uniqInsert(String element)
//Checks for element in linked list and if present returns false
// if it is not present inserts element
{
  LLStringNode node;
  node = log;

  while (node != null)
  {
    if (element.equalsIgnoreCase(node.getInfo()))  // if they match
      return false;
    else
      node = node.getLink();
  }
  LLStringNode newNode = new LLStringNode(element);
  newNode.setLink(log);
  log = newNode;
  return true;
}
```

Implementation of uniqInsert() above works fine and was good practice in traversing a linked list. However, calling the authors' contains() and insert() methods would have been easier and more efficient from a development perspective.

Feedback continues on next page.

```java
//new smallest() method by Alec

//PRECONDTIONS:String is not empty
public String smallest()
//returns smallest(in terms of lexicographic ordering) string in String log
{
  LLStringNode node;
  node = log;
  String smallest = node.getInfo();
  node = node.getLink();
  while(node!= null)
  {
    System.out.println(node.getInfo() + ", compare to current smallest: " + node.getInfo().compareTo(smallest));
     if(node.getInfo().compareTo(smallest)<= 0){
       smallest = node.getInfo();
     }
     node = node.getLink();
  }
  return smallest;
}
```

Moving to the next node prior to the start of the loop makes code slightly more efficient; no need to test **smallest** vs. the first node in the list.