Due Sunday March 1st, 11.55pm

This project will use the BRIDGES API with an actor/movie dataset from IMDB. For this, you will adapt your classes from part 1 to use BRIDGES elements, which will enable you to visualize the data structure you are constructing and testing. ***Note that the visualization will also be a means to debug your program, as it will let you view your data structure and its contents after each operation.***

The BRIDGES API is accessible from the Moodle pages, or from **this link.**

Dataset:

The actor/movie dataset, curated from IMDB is a text file containing a set of actor/movie pairs, one pair per line. **Note that both actors and movies can and will appear multiple times**, in the data file.

Project Tasks:

1. You will modify your singly linked list interface from your previous project, so as to use the Bridges elements. Use the Hello World tutorial as a guideline. In particular, the Link class will now use the equivalent Bridges class to implement each node of the linked list. Once you have the interface working, then run your driver and generate example lists of a few nodes and ensure the visualization looks right.

2. **List Construction:** As detailed above, you will write functions to read in the actor/movie pairs from a data file. **You will build a list of actors ordered by their names(both first and last names have been concatenated, so simply use their full name to order the records).** Note that the actor name is unique. The corresponding movie will become part of a string, and maintained in the 'data' part of the node. Each insertion of an actor into the sorted linked list will involve searching for that actor in the list: if the actor is found, the corresponding movie is appended to the movies' string in that node(use carriage returns between movies for good formatting); if it was not found, then a new node is created and inserted at the correct location in the list, to maintain the sorted order.

   - Once the list has been fully constructed, highlight the head, current node and the tail of the list.
   - Mousing over a node should indicate the movies corresponding to each actor.
   - Keep a running count the number of operations(number of nodes traversed) to insert each actor. Youw will print the average number of nodes traversed at the end.
   - **Note:** In the singly linked list, current points to the following node, else deletions are problematic. Thus, **your highlighting should be the node next to the current node, which is what is being operated upon in normal list operations.**

3. You will support 3 operations on your list and accept these as standard input from console:

   - **Find An Actor:** Your program should accept an actor actor name. If the actor is found, then highlight that actor in the list in a unique color and update the viualization . Change the list's current position to this node.

- **Remove an Actor:** This simply removes the actor at the current position. Again, update the visualization and the current position (which is highlighted).
- **Add an actor movie pair:** Add an actor/movie pair to the list. Update the current position and highlight this new actor. Note that the incoming actor may already be in the list, in which case the movie list will be updated(current position moves to this node).

Implementation Details. You can augment the LList class with the needed functions to implement the above tasks. Make sure that you maintain the generic implementation. In this project, your generic parameter (E) will be a string(to contain the movies). Implementations that violate this boundary will lose points.

Evaluation:

An interactive demonstration of the project to the TA will be required. Test yor program thoroughly so it can perform the above tasks. Submit your sources to Moodle.