Data Structures (CS 1520) Homework #3 Due: 9/29/12 (Sat.) at 11:59 PM

Objective: To gain experience using a cursor-based list by implementing a text-editor program. Plus, it provides experience using a doubly-linked list with header and trailer nodes.

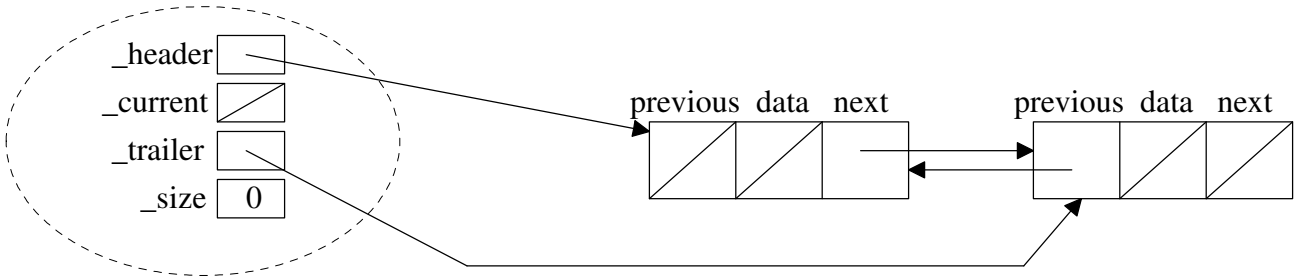
To start the homework: Download and extract the file hw3.zip from
http://www.cs.uni.edu/~fienup/cs1520f12/homework/index.htm
The hw3.zip file contains:

- the Node class (in the node.py module) and the Node2Way class (in the node2way.py module)
- the skeleton CursorBasedList class (in the cursor_based_list.py module) which you will complete
- the cursorBasedListTester.py file that you can use to interactively test your CursorBasedList class.

Part A: Recall that in a **cursor-base list** a *cursor* (indicating the *current item*) can be moved around the list with the cursor being used to identify the region in the list to be manipulated. We will insert and removing items relative to the current item. A *current item* which is always defined as long as the list is not empty.

Cursor-based operations	Description of operation
`L.getCurrent()`	Precondition: the list is not empty. Returns the current item without removing it or changing the current position.
`L.hasNext()`	Precondition: the list is not empty. Returns True if the current item has a next item; otherwise return False.
`L.next()`	Precondition: hasNext returns True. Postcondition: The current item has moved right one item
`L.hasPrevious()`	Precondition: the list is not empty. Returns True if the current item has a previous item; otherwise return False.
`L.previous()`	Precondition: hasPrevious returns True. Postcondition: The current item has moved left one item
`L.first()`	Precondition: the list is not empty. Makes the first item the current item.
`L.last()`	Precondition: the list is not empty. Makes the last item the current item.
`L.insertAfter(item)`	Inserts item after the current item, or as the only item if the list is empty. The new item is the current item.
`L.insertBefore(item)`	Inserts item before the current item, or as the only item if the list is empty. The new item is the current item.
`L.replace(newValue)`	Precondition: the list is not empty. Replaces the current item by the newValue.
`L.remove()`	Precondition: the list is not empty. Removes and returns the current item. Making the next item the current item if one exists; otherwise the tail item in the list is the current item unless the list in now empty.

The `cursor_based_list.py` file contains a skeleton `CursorBasedList` class. You MUST uses a doubly-linked list implementation with a *header* node and a *trailer* node. An empty list looks like:



All "real" list items will be inserted between the header and trailer nodes to reduce the number of "special cases" (e.g., inserting first item in an empty list, deleting the last item from the list).

Use the provided `cursorBasedListTester.py` program to test your list.

Data Structures (CS 1520) Homework #3 Due: 9/29/12 (Sat.) at 11:59 PM

Part B: You are to write a simple text-editor program that utilizes your `CursorBasedList` class. (You might want to start with the `cursorBasedListTester.py` program as a rough starting point.) Your text-editor program should present a menu of options that allows the user to:

- enter a filename of a text (.txt) file to edit. Your program should then insert each line from the text file into a cursor-based list.
- create a new text (.txt) file to edit. Your program should create an empty cursor-based list.
- navigate and display the first line, i.e., the first line should be the current line
- navigate and display the last line, i.e., the last line should be the current line
- navigate and display the next line, i.e., the next line should become the current line. If there is no next line, tell the user and don't change the current line
- navigate and display the previous line
- insert a new line before the current line
- insert a new line after the current line
- delete the current line and have the line following become the current line. If there is no following line, the current line should be the last line.
- replace the current line with a new line
- save the current list back to a text file

Implement AND fully test your text-editor program. Part of your grade will be determined by how **robust** your text-editor runs (i.e., does not crash) and how **user-friend**/intuitive your program is to use. You are required to submit a brief User's manual on how to use your text-editor.

For extra credit, your program may also do one or more of the following:

- Provide additional text-editor functionality: Find word, Replace word, Copy a line, Paste a line, etc. Be sure to include these additional features in your User's manual.
- Use a GUI package (e.g., Tkinter discussed at http://docs.python.org/py3k/library/tk.html and http://infohost.nmt.edu/tcc/help/pubs/tkinter/) instead of a text-menu interface. Note: you should still use the cursor-based-list ADT to store the lines of text as the editor runs.

Submit all necessary files (userManual.doc, cursor_based_list.py, node.py, node2way.py etc.) with your text-editor program file(s) as a single zipped file (called hw3.zip) electronically at

https://www.cs.uni.edu/~schafer/submit/which_course.cgi