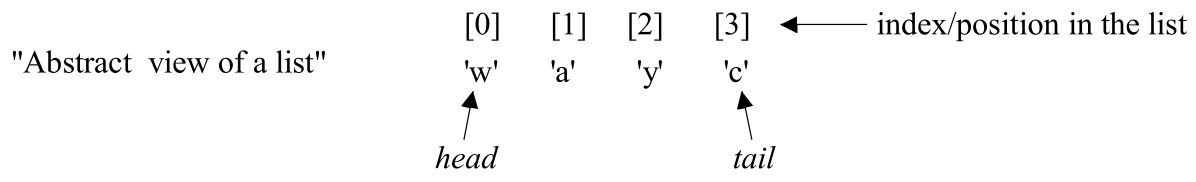
Data Structures (CS 1520) Programming Project 4

Objective: To gain experience implementing linked data structures by implementing a cursor-based list using doubly-linked nodes.

To start the homework: Download and extract the file project4.zip.

The project4.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.

<u>**Background:**</u>
A "list" is a generic term for a sequence of items in a linear arrangement. Unlike stacks, queues and deques access to list items is not limited to either end, but can be from any position in the list. The general terminology of a list is illustrated by:



There are three broad categories of list operations:
- **index-based operations** - the list is manipulated by specifying an index location, e.g.,
  myList.insert(3, item) # insert item at index 3 in myList
- **content-based operations** - the list is manipulated by specifying some content (i.e., item value), e.g.,
  myList.remove(item) # removes the item from the list based on its value
- **cursor-base operations** - a *cursor* (current position) can be moved around the list, and it is used to identify list items to be manipulated, e.g.,

```
myList.first()      # sets the cursor to the head item of the list
myList.next()       # moves the cursor one position toward the tail of the list
myList.remove()     # deletes the second item in the list because that's where the cursor is currently located
```

The following table summarizes the operations from the three basic categories on a list, L:

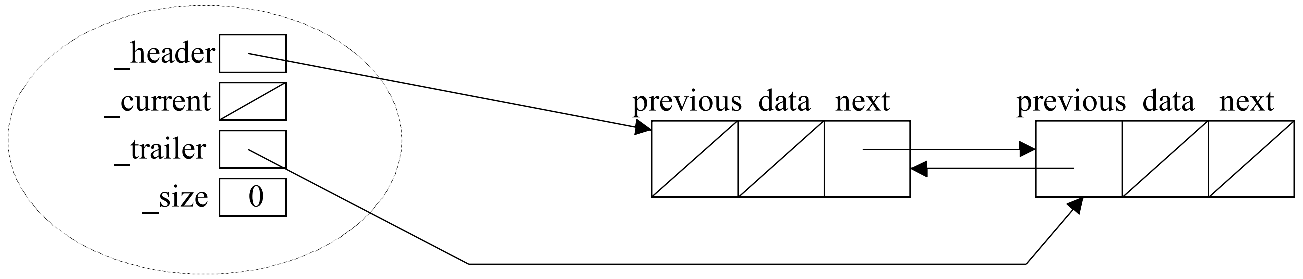
Index-based operations	Content-based operations	cursor-based operations
`L.insert(index, item)`	`L.add(item)`	`L.hasNext()`
`item = L[index]`	`L.remove(item)`	`L.next()`
`L[index] = newValue`	`L.search(item) #return Boolean`	`L.hasPrevious()`
`L.pop(index)`	`i = L.index(item) #return index of item`	`L.previous()`
		`L.first()`
		`L.last()`
		`L.insert(item)`
		`L.replace(item)`
		`L.remove()`

Built-in Python lists are unordered with a mixture of index-based and content-based operations. We know they are implemented using a contiguous block of memory (i.e., an array). The textbook talks about an unordered list ADT, and a sorted list ADT which is more content-based. Both are implemented using a singly-linked list.

For this project we will use a **cursor-base list.** A *cursor* (indicating the *current item*) can be moved around the list (via next, previous, first, last operations) with the cursor being used to identify the region of the list to be manipulated. We will insert and removing items relative to the current item. A *current item* must always be defined as long as the list is not empty. Below is a complete list of **Cursor-based operations** and their descriptions.

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.

<u>Part A:</u> CursorBaseList implementation
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. **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, etc.). An empty list looks like:



Complete the `CursorBasedList` class and use the provided `cursorBasedListTester.py` program to test your list.

<u>PART B:</u> Text-editor program description
Once you have your `CursorBasedList` class finished, you are to write a simple text-editor program that utilizes your `CursorBasedList` class.

When your text-editor program starts, it should ask for a text-file name (`.txt`) to edit. If the file name exists, it should load the file into an initially empty `CursorBasedList` object by reading each line from the file and use the `insertAfter` method to append the line to the list. **Each node in the list will hold a single line of the text**

file. If the text-file name specified at startup does not exist, an empty `CursorBasedList` object is created to model editing a new file.

Regardless of whether you loaded a file or just created an empty list, a menu-driven loop very similar to the `cursorBasedListTester.py` program should allow you to edit the file's content by modifying the list. You should NOT need to modify your `CursorBasedList` class only create a `CursorBasedList` object and use its methods. **Make sure that your editor does not violate any preconditions of the `CursorBasedList` methods, so your editor is robust, i.e., does not crash when editing.**

When done editing, the lines of data contained in the nodes of the `CursorBasedList` are written back to the text file.

Your text-editor program should present a menu of options that allows the user to:

- 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. Similarly, if there is no previous line, tell the user and don't change the current 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

Warning: When you load a text file into your list nodes, you can leave the '\n' characters on the end of each line of text. However, remember to add a '\n' character to the end of inserted lines or replacement lines.

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 provide do one or more of the following additional text-editor functionality:

- Find word and Find next occurance
- Replace a specified word/string on the current line by another word/string
- Copy and Paste a line, etc.

Be sure to include these additional features in your User's manual.

Submit all program files (cursor_based_list.py, node.py, node2way.py, text_editor.py, etc.) and User's manuel as a single zipped file (called project4.zip) to the Programming Project 4 Submission link.