Data Structures (CS 1520) Lab 6 Name:________________

Objective: Gain experience implementing a linked data structure by using a doubly-linked list to implement a positional list ADT.

<u>Part A</u>: The position-base operations described in Tables 16.4 and 16.5 of the Lambert textbook treats the cursor poorly. For example, the `remove` operation is described as:

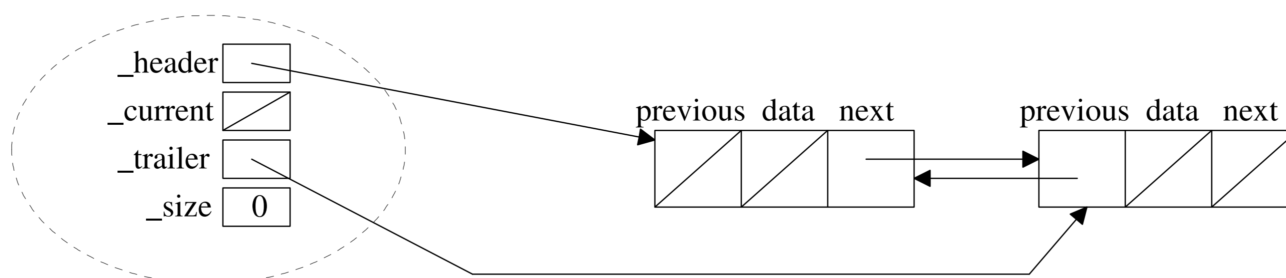
"*Precondition*: There have been no intervening `insert` or `remove` operations since the most recent `next` or `previous` operation. Removes the item returned by the most recent `next` or `previous`."

Typically, if a method has a precondition, then there is another method to check the precondition. For example, the `next` operation has a precondition: "`hasNext` returns `True`." However, the `remove` operation's precondition cannot be checked without the user application maintaining a list of preceeding operations -- a bad design! The problem occurs because the `insert` and `remove` operations leave the cursor in an undefined state.

Instead of thinking of a cursor between two list items, let's have a *current item* which is always defined as long as the list is not empty. We will insert and delete relative to the current item.

Positional-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 `mypositionalList.py` file contains a `LinkedPositionalList` class, which uses a doubly-linked list with a *header* node and *trailer* node to reduce the number of "special cases" (e.g., inserting first item in an empty list). An empty list looks like:



Use the `testList.py` program to test your list.

After implementing and testing your LinkedPositionalList class, raise you hand and demonstrate your code.