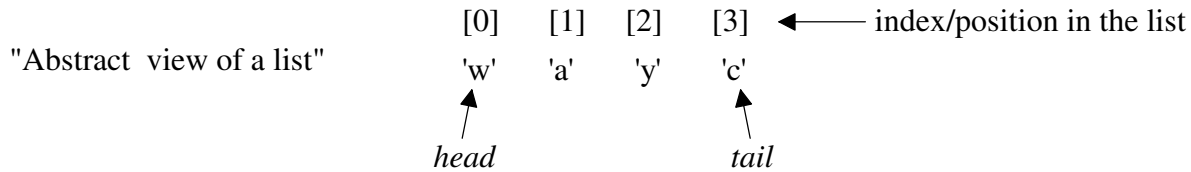


Background: Read sections 13.5, 13.6 and chapter 16 from the Lambert text. A “list” is a generic term for a sequence of items in a linear arrangement. Unlike stacks and queues, access to list items is not limited to either end, but can be from an position in the list. The general terminology of a list is illustrated by:



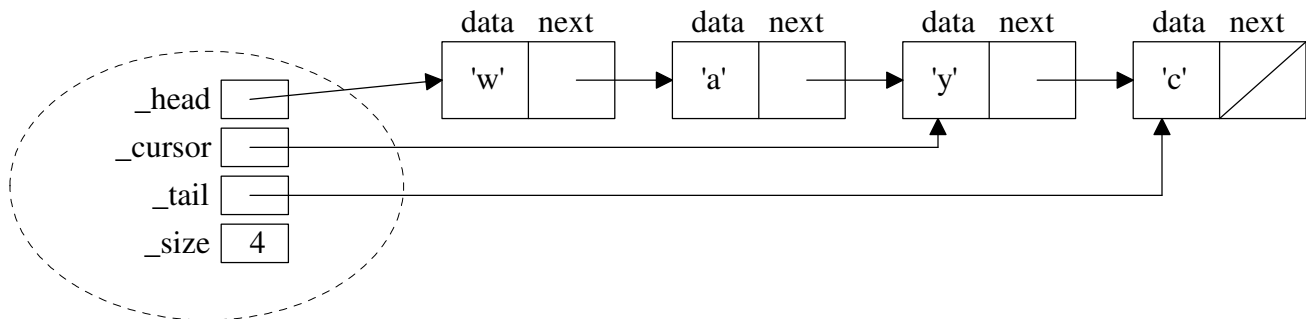
There are three broad categories of list operations defined in the textbook:

- **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) in the list, e.g.,
`myList.index(item)` # search for the item in the list and return its index if found; otherwise return -1
- **positional-base operations** - a *cursor* (current position in the list) 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 of the list
`myList.remove()` # deletes the first item in the list because that’s where the cursor is located

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

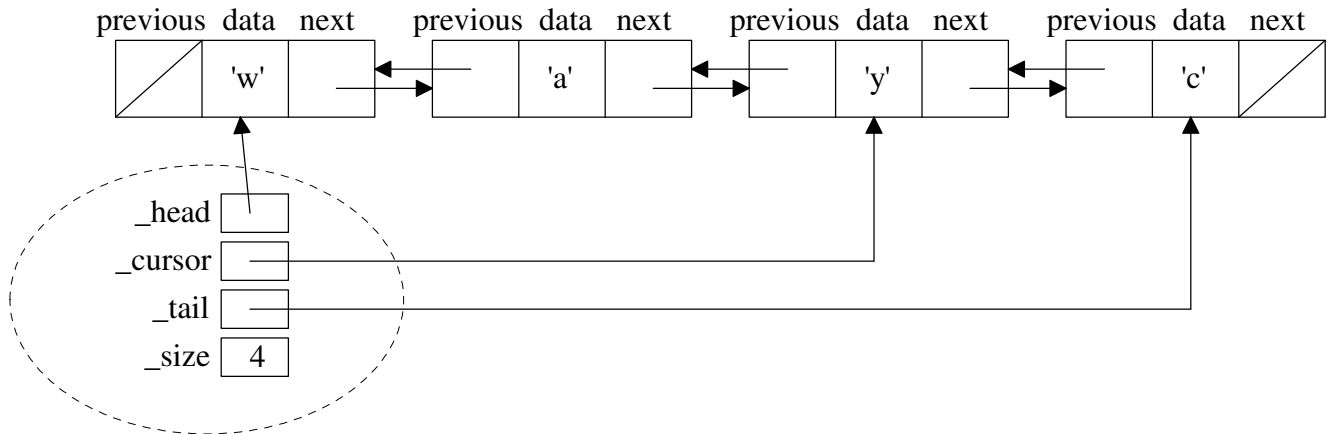
Index-based operations	Content-based operations	Positional-based operations
<code>L.insert(index, item)</code> <code>item = L[index]</code> <code>L[index] = newValue</code> <code>L.remove(index)</code>	<code>L.append(item)</code> <code>L.index(item)</code>	<code>L.hasNext()</code> <code>L.next()</code> <code>L.hasPrevious()</code> <code>L.previous()</code> <code>L.first()</code> <code>L.last()</code> <code>L.insert(item)</code> <code>L.replace(item)</code> <code>L.remove()</code>

1. Why would a singly-linked list be a bad choice for implementing a positional-based list ADT?



2. Suppose we had a doubly-linked list implementation for a positional-based list ADT:

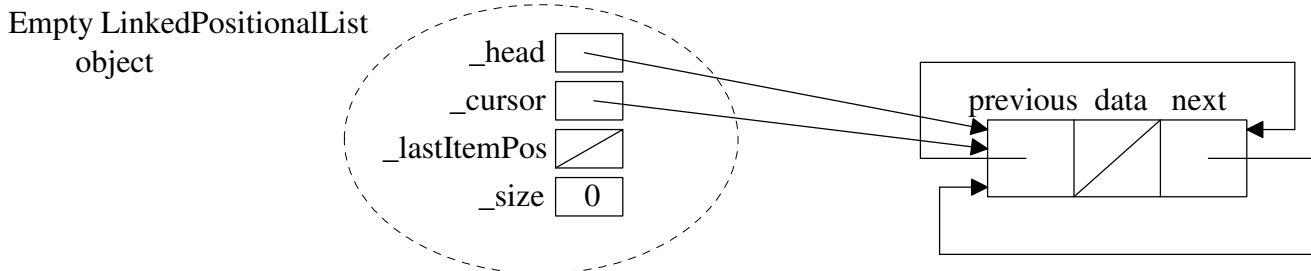
```
class TwoWayNode(Node):
    def __init__(self, data, previous = None, next = None):
        Node.__init__(self, data, next)
        self.previous = previous
```



a) `L.insert(item)` If the cursor is defined, insert `item` after it; otherwise, insert `item` at the end of the list. Write the “normal” case (cursor defined and inserting between two existing nodes) code.

b) In addition to the “normal” case (inserting between two existing nodes), what special cases would the `insert` operation need to handle for the doubly-linked list implementation above?

c) The textbook’s `LinkedPositionalList` class, uses a circular, doubly-linked list with a *sentinel* (or *header*) node to reduce the number of “special cases”. An empty list looks like:



Write the complete `insert` operation code.