CSCI-SHU 210 Data Structures

Recitation 7 Linked Lists

1. Implement Stack using Single ended singly linked list

Let us continue from Tuesday's class. This time we are going to use a singly linked list to store stack's data.

Before we implement, let's take a comparison between different stack implementations:

	circular array	List append/pop	Single ended singly linked list
Push()	O(1) amortized	O(1) amortized	O(1)
Pop()	O(1) amortized	O(1) amortized	O(1)

```
Your task: Implement class LinkedStack. Including those functions:
top(self)
push(self, e)
pop(self)
unOrderedSearch(self, target)
printAll(self)
```

2. Implement Queue using Double ended singly linked list

```
Your task 1: Why queue needs a Double ended singly linked list instead of Single ended singly linked list?
```

Before we implement, let's take a comparison between different queue implementations:

	circular array	List append/pop	Double ended singly linked list
enqueue()	O(1) amortized	O(1) amortized	O(1)
dequeue()	O(1) amortized	O(n)	O(1)

```
Your task 2: Implement class LinkedQueue. Including those functions:
***Although you can use either side of linked list as front of queue, we
   choose head side as front.
```

```
first(self)
dequeue(self)
enqueue(self, e)
__str__(self)
```

3. Implement Deque using `Double ended double linked list`

```
Your task: Implement class LinkedDeque. Including those functions:

*** To save time, some of the following functions are already implemented.
first(self)                    Equivalent to LinkedQueue.first()
last(self)
add_first(self, e)
add_last(self, e)              Equivalent to LinkedQueue.enqueue()
delete_first(self)             Equivalent to LinkedQueue.dequeue()
delete_last(self)
print_reverse_order(self)
```

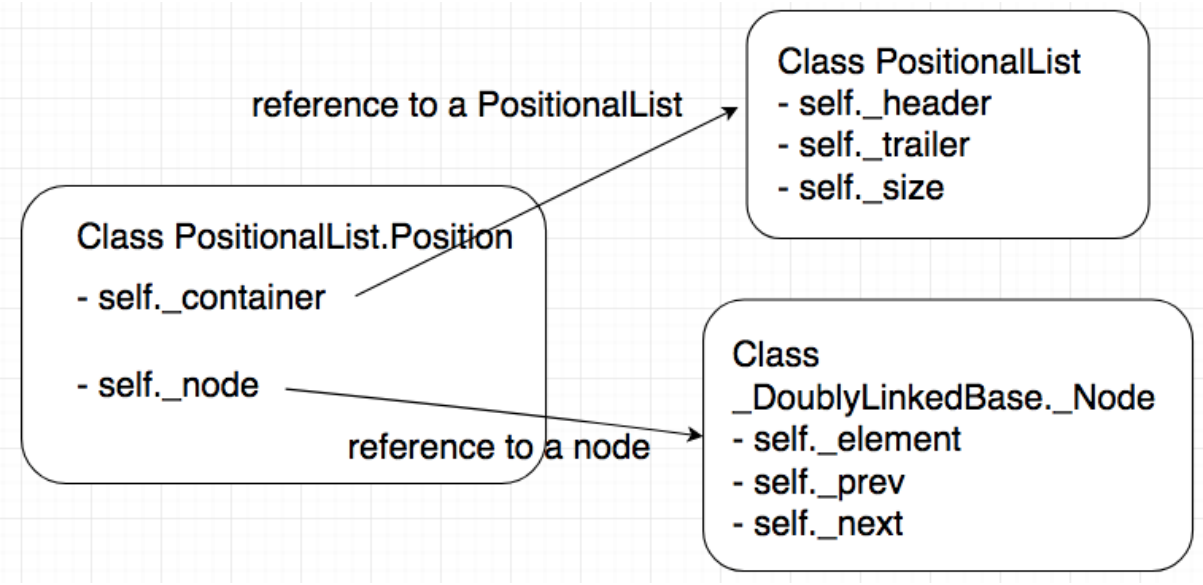
4. Positional list

Now let's pay attention to Positional lists. This is a special doubly linked list class that performs validation before insertion.

Take a look at the following code snippet:

```
1. myList = PositionalList()
2. myList.add_after(some_node, 9000)   # Uh-oh
```

When we call `add_after()`, we are trying to insert value 9000 after `some_node`. How do you make sure, `some_node` belongs to `myList`? The positional list wraps the actual node with another class called PositionalList.Position, so each node not only have references to `prev` and `next` node, but also have reference to the entire linked list. Illustrated by the following diagram:



Now I have an additional reference to the linked list I belong to, I can perform validation before insertion. Making sure I'm inserting in the correct linked list, not some other linked list that I don't belong to.

Thus, I can perform node validation like this:

```
def _validate(self, p):
    """Return position's node, or raise appropriate error if invalid."""
    if not isinstance(p, self.Position):
        raise TypeError('p must be proper Position type')
    if p._container is not self:
        raise ValueError('p does not belong to this container')
    if p._node._next is None: #convention for deprecated nodes
        raise ValueError('p is no longer valid')
    return p._node
```

Your task 1: Take a look at class PositionalList. Try the test code, then type some test code on your own. Try to understand this class' structure.

Your task 2: Complete function merge(self, other).

This function adds other Positional list to the end of self Positional list.