SOFTWARE 1 PRACTICAL

LINKED LISTS & STACKS & QUEUES

Week 10 – Practical 9

Exercise 1: *linked list implementation of a stack*
During this week lecture, you have seen how to implement a list using `ListNode` objects. We want to reuse these objects to implement a stack (Last-In-First-Out). However, we want to encapsulate these within a class named `LinkedStack`, within the module `linkedstack`, which has an attribute **`_top`**, which references the top of the stack (a ListNode) in the same way as **head** referenced the front of a linked list, and an attribute **`_size`** to store the number of elements in the stack. Implement the following methods:

1. `__init__(self)` that creates an empty stack, that is **_top** reference **None**.
2. `push(self, value)` that pushes the value at the top of the stack.
3. `__str__(self)` that return a string representing the stack in the form:

   `'LinkedStack([top_value, …, bottom_value])'`
4. `pop(self)` that removes and returns the value at the top of the stack. The method should raise a `ValueError` if the stack is empty.
5. `peek(self)` which returns (but does not remove) the value at the top of the stack.
6. `__len__(self)` that returns the number of values in the stack.
7. `isempty(self)` that returns `True` if the stack is empty, `False` otherwise.

Exercise 2: *linked list implementation of a queue*
Similarly, we want to implement a Queue ADT using a linked list implementation. First you should determine what attribute(s) is(are) needed for the class. Remember, a queue is a First-In-First-Out container, which means elements are removed from the front of the queue and are added to the back of the queue. In addition, adding to/removing from a queue must be fast. Once you are satisfied with your design, Implement the following methods:

1. `__init__(self)` that creates an empty queue. Determine what an empty queue should look like.
2. `__str__(self)` that returns a string representing the stack in the form:

   `'LinkedQueue([front_value, …, back_value])'`
3. `enqueue(self, value)` that enqueues the value at the back of the queue.
4. `pop(self)` that removes and returns the value at the front of the queue. The method raises a `ValueError` if the stack is empty.

5. `peek(self)` which returns (but does not remove) the value at the front of the queue.
6. `__len__(self)` that returns the number of values in the queue.
7. `isempty(self)` that returns `True` if the queue is empty, `False` otherwise.

Exercise 3:

In this exercise, you will continue the development of the class LinkedList provided during the lecture (download the file from the VLE).

- `clear(self)`: Remove all items from list.
- `index(self, value, start=0, stop=2147483647)`: Return first index of value. Raises `ValueError` if the value is not present.
- `insert(self, index, object)`: Insert object before index, raises `IndexError` if index is out of range.
- `remove(self, value)`: Remove first occurrence of value. Raises `ValueError` if the value is not present.

Overload the following operators:

- `__getitem__(self, index)`:
  x.__getitem__(y) <==> x[y], raises `IndexError` if index is out of range.
- `__setitem__(self, index, value)`:
  Set self[index] to value, raises `IndexError` if index is out of range.
- `__contains__(self, value)`: Returns value in self in the same way we write the statement:

```
4 in [1, 2, 4].
```