CSC148, Lab #6
week of October 21st, 2013

This document contains the instructions for lab number 6 in CSC148H1. To earn your lab mark, you must actively participate in the lab. We mark you in order to ensure a serious attempt at learning, **not** to make careful critical judgments on the results of your work.

General rules

We will use the same general rules as for the first lab (including pair programming). See the instructions at the beginning of Lab 1 to refresh your memory.

Overview

In this lab you will begin to work with linked lists. Here are some general hints on how to proceed:

- **Draw lots of pictures** to get a very clear idea of exactly what the linked structure should look like before, during, and after each operation you are trying to implement. This is important: if you skip this, you are likely to have only a vague idea of what it is your code should do, and you are much more likely to make mistakes!
- Think carefully about what variables and attributes each part of your picture represents. If you understand this, then you will be able to tell exactly what changes you need to make in order to carry out the task you want to carry out.

re-implement Queue

(Student `s1` drives, student `s2` navigates)

In a file named `linked_queue.py`, write code for a Node class (take a look at the lecture materials from this week to see how we did this in class).

Next, write a class `Queue` that implements the Queue ADT with a linked list (see csc148queue.py for the definition of the ADT and of the methods you should implement). Do not use any list or dictionary or other standard container! Remember: the point of this exercise is to practice linked lists.

Use the file testqueue.py to test that your implementation seems to work correctly. Once you have done so, use the file timequeue.py to check the efficiency of your implementation. If you have done it correctly, `timequeue.py` should show that the time is the same no matter how many elements are in the queue (what did your queue implementation based on Python `list` do?)

Here are some hints:

Hint #1: Instances of your Queue class should store two attributes: a reference to the first Node in the linked list storing the Queue elements, and a separate reference to the last Node in the linked list.

Hint #2: You need to be careful to update your attributes correctly when the queue starts out (or ends up) empty.

Once you are done and confident that your code works, show your work to your TA.

implement `list`, (sort of)

(Student `s1` drives, student `s2` navigates). If there is time, tackle the following task.

You know that in a linked list, elements are stored in a sequence of Node objects, where each Node stores one list element and a reference to the next Node in the sequence. But a `LinkedList` class is a higher-level structure that stores the entire chain of Nodes (indirectly, by keeping a single reference to the first Node), along with methods that perform list-like operations on the entire list.

Create a new file named `linkedlist.py`.

In this file, write a "private" class `_Node` that stores one object reference and one `_Node` reference (called `link`). (The underscore at the start of the name makes this class "private", which means that it should not be used from outside the file linkedlist.py.)

Next, write a class `LinkedList`, that stores one reference to a `_Node` (the first one in the list) and one int (the number of elements in the list). Your constructor should take no extra argument (in addition to the usual self) and it should initialize the attributes to make the list empty. In other words, a call to your constructor would look something like:

```
list0 = LinkedList()
```

Your class should implement the following methods (try to complete at least a few):

```
__len__(self):
    Return the number of elements in this linked list.
__contains__(self, elem):
    Return True iff this linked list contains elem.
__getitem__(self, ind):
    Return the element stored at index ind in this linked list.
    Raises IndexError if ind < 0 or ind > size1.
insert(self, elem):
    Insert elem at index 0 in this linked list.
    Does not overwrite any element previously at index 0,
    though the index of that element (and all that follow) increases by one.
```

Some of these are special methods (denoted by the double-underscore `__`), that are never called directly, but that get called automatically in appropriate situations. For example, here is one way that your class could get used and its expected behaviour:

```
list1 = LinkedList()  # creates a new empty linked list
print(len(list1))  # automatically calls list1.__len__(); prints '0'
print(5 in list1)  # automatically calls list1.__contains__(5); prints 'False'
print(list1[0])  # automatically calls list1.__getitem(0); raises IndexError
list1.insert(15)
list1.insert(17)
```

```
print(len(list1)) # prints ’2’
print(list1[0]) # prints ’17’
```