FIT1008 Introduction to Computer Science (FIT2085 for Engineers)

Tutorial 8

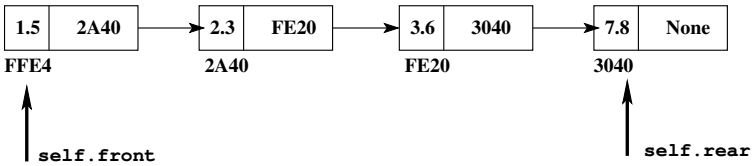
Semester 1, 2019

Objectives of this tutorial

- To understand programming with linked data structures.

Exercise 1 *

Consider the `Queue` class implemented with linked nodes provided in the lectures. Consider an object `my_queue = Queue()` of this class, to which we have appended four floats (1.5, 2.3, 3.6, and 7.8), and has the current form:



where the hexadecimal number under each node represents the address of the node. Without running the code (that is, by looking at it and tracing it by hand), write down the values of the following variables after executing `item = my_queue.serve()`:

- `my_queue.front`
- `my_queue.rear`
- `item`
- `my_queue.front.link`

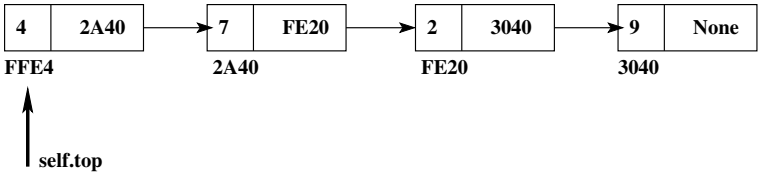
Exercise 2 *

Consider the `Stack` class implemented with linked nodes given in the lectures. Consider adding a method `sum_all` to the class whose function is to return the sum of all elements in the stack (zero if the stack is empty) without modifying the stack itself. The following shows three possible implementations of such method:

```
1  def sum_all(self):
2      sum = 0
3      while (not self.is_empty()):
4          sum += self.top.item
5          self.top = self.top.link
6      return sum
7
8  def sum_all(self):
9      current = self.top
10     sum = 0
11     while current.link is not None:
12         sum += current.item
13         current = current.link
14     return sum
15
16 def sum_all(self):
17     current = self.top
18     sum = 0
19     while current is not None:
20         current = current.link
```

```
21            sum += current.item
22        return sum
```

Consider an object `my_stack` with the form



Without running the code (that is, by looking at it and tracing it by hand), show the effect on the list of calling `result = my_stack.sum_all()` for each definition above. Show also the final value of `result`. Provide a correct definition for the above method.

Exercise 3 *

The following table compares the main advantages, for time efficiency, of using arrays (contiguous memory) and of using linked data structures for implementing data types. Some of these advantages are however only important for some, not all, data types.

Discuss which of these advantages (if any) are not important for implementing stacks.

Array (contiguous memory)	Linked nodes
Constant time access to any element	No shuffling required
Easy/fast traversal backwards	Easy/fast re-sizing (up and down)

Exercise 4 *

Consider a queue data type provided by the `Queue` class which is implemented using some data structure (you do not need to know which one) and defines the following usual methods for queues:

```
serve()
append(new_item)
is_empty()
```

Write a Python function `def interleave(queue1, queue2)` that returns a new queue that interleaves the two queues in a fair way, i.e., the new queue has at the front the first element of `queue1` then the first element of `queue2`, then the second element of `queue1`, then that of `queue2`, and so on, until one of the queues is empty, in which case the rest of the elements in the other queue are appended at the end. For example, if `queue1` is (from front to rear) 20 -1 2 10 7 and `queue2` is 3 6 -4, then the new queue should be 20 3 -1 6 2 -4 10 7. If `queue1` is 20 -1 2 10 7 and `queue2` is empty, then the new queue should be 20 -1 2 10 7. And if both `queue1` and `queue2` are empty, the new queue is also empty.

Note that you are using the `Queue` class, not implementing it. Therefore, you have no idea (nor should you care) about how it has been implemented. Note also that you are only allowed to use the above methods to access/modify the queue (i.e., no other methods are allowed for queues). Also, you are allowed to modify the input queues `queue1` and `queue2`.

Exercise 5

Show how to implement the Queue ADT using two stacks (and their corresponding ADT). What is the worst-case running time of the `serve` operation? What is the average running time?