

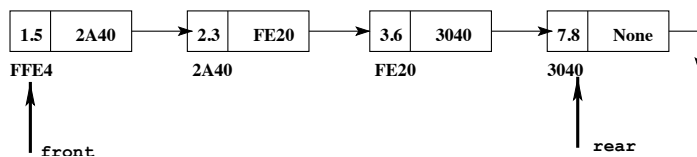
*FIT1008 – Intro to Computer Science*  
*Tutorial 10*  
Semester 1, 2017

*Objectives of this tutorial*

- To understand how linked queues and linked lists work.
- To understand how Iterators work.

**Exercise 1**

Consider a class for a linked queue defined in the way given in the lecture notes. Consider an object `my_queue` of this class that has the current form:



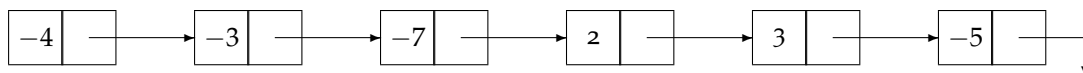
where the hexadecimal number under each node represents the address of the node.

After the call `item = my_queue.serve()`; what is the value of:

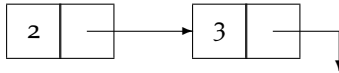
- `my_queue.front`
- `my_queue.rear`
- `item`
- `my_queue.front.next`

**Exercise 2**

Suppose `List` class implements a linked list. Add a method `delete_negative` to the class that, when the list contains numbers, eliminates from the list any node containing a negative element. For example, given the linked list:



where -4 is at the head of the list, a call to `the_list.delete_negative()` would leave the list as:



where 2 is at the head.

### Exercise 3

Suppose List class implements a linked list and consider the following function that accepts two of those lists:

```

1 def mystery(a_list1, a_list2):
2     if a_list1.head is None:
3         a_list1.head = a_list2.head
4     else:
5         current1 = a_list1.head
6         current2 = a_list2.head
7         while current2 is not None:
8             temp = current1.next
9             current1.next = current2
10            if current2.next is None:
11                current2.next = temp
12            else:
13                current1 = current2.next
14                current2 = temp
  
```

- (i) Consider a call to `mystery(a_list1, a_list2)` where `a_list1` is a list with elements 1,2,5,3,8 (in that order) and `a_list2` is a list with elements 0,9,7,4,6,1,0,5 (in that order). Draw the memory diagram for the resulting lists before and after the function is executed and explain what the function does.
- (ii) What is the best and worst Big O time complexity of our `mystery` function in terms of the lengths  $n_1$  and  $n_2$  of the lists? Given an explanation.

### Exercise 4

Consider an List class that defines a list data type with the following methods:

```

__init__(self)           # for instance creation via List()
__iter__(self)           # returning an instance of Iterator via iter()
  
```

where Iterator is an iterator class of List that provides the method:

```

__next__(self)           # for accessing the next item via next()
  
```

- (a) Define a method `appears(a_list, item, k)` where `k` is an integer and `appears` returns `True` if the item appears at least `k` times in `a_list`. For example, given a list with elements 8, 7, 6, 7, 8, 6, 5, 3, 6, 100, where element 8 is the head, a call to `appears(a_list, 6, 3)` would return `True`, while a call to `appears(a_list, 6, 4)` would return `False`.

IMPORTANT: the implementation should be such that you stop as soon as you know the answer is `True`.

- (b) What is the best/worst Big O complexity for a list of `N` elements? Explain.

### *Exercise 5*

Write an iterator for a Circular Queue, that iterates through all the items in the queue from front to rear.

### *Exercise 6*

Write an iterator for prime numbers less than `max`, which iterates through the prime numbers in increasing order from 2 to the largest prime less than `max`.