DNHI Homework 4
Searching and Sorting

Problem 1

Given the IntegerQueue ADT below state the return value and show the content of the, initially empty, queue of 'Integer' objects after each of the following operations. If any of the operations below is invalid or would cause program to crash, state it and explain what is wrong with it (then proceed with the next step ignoring the invalid line). Assume that 'queue' is a reference of type 'IntegerQueue'.

Informal Specification   The 'IntegerQueue' contains a (possibly empty) collection of objects of type 'Integer'. The queue supports the following operations:

'insert'/'enqueue' This operation adds a 'Integer' object, given as a parameter, to the end of the queue of integers.

'remove'/'dequeue' This operation removes and returns a Integer object from the front of the queue of integers. If the queue is empty, null should be returned.

'toString' This operation produces a 'String' that contains all 'Integer' objects stored in the queue from font to back separated by a single comma and a single space. If the queue is empty, an empty string should be returned.

```
    queue.insert( 15 );
    queue.insert( 3 );
    queue.insert( -15 );
    queue.insert( 35 );
    queue.remove();
    queue.remove();
    queue.remove();
    queue.insert( 13 );
    queue.remove();
    queue.remove();
    queue.remove();
    queue.insert( 3 );
```

A   Assume that the implementation is array based and follows the efficiency ideas that we discussed in class. Assume that the initial capacity of the array to store the queue is equal to 4 and that its size is doubled whenever the array runs out of room.

B   Assume that the implementation is reference based and the head reference points of the front of the queue.

C   Discuss efficiency/performance of each the 'insert' and 'remove' operations for each implementation (for the reference based implementation, consider the difference in performance if you have an additional reference to the end of the queue).

Problem 2

Given the 'CharStack' ADT that we discussed in class show the content of the, initially empty, stack of 'Character' objects after each of the following operations. If any of the operations below is invalid or would cause program to crash, state it and explain what is wrong with it. Assume that stack is a reference of type 'CharStack'.

Informal Specification   The 'CharStack' contains a (possibly empty) collection of objects of type 'Character'. The stack supports the following operations:

'insert'/'push' This operation adds a 'Character' object, given as a parameter, to the top of the stack of characters.

'remove'/'pop' This operation removes and returns a 'Character' object from the top of the stack of characters.

'peek' This operation returns a 'Character' object from the top of the stack of characters.

'toString' This operation produces a meaningful 'String' representation of the stack.

```
    stack.push('c');
    stack.push( new Character('s') );
    stack pop();
    char p = 's';
    stack.push( p );
    stack.push( p );
    stack.push( new Character('1') );
    stack.peek();
    stack.pop();
    stack.push('%');
    stack.peek();
    stack.push('A');
    stack.push('X');
    stack.pop();
    stack.pop();
```

A   Assume that the implementation is array based and follows the efficiency ideas that we discussed in class. Assume that the initial capacity of the array to store the stack is equal to 4 and that its size is doubled whenever the array runs out of room.

B   Assume that the implementation is reference based. Where should the 'head' reference point to (bottom of the stack or top of the stack).

C   Discuss efficiency/performance of each the 'insert'/'push' and 'remove'/'pop' operations for each implementation.

Problem 3

The 'GenericList' interface and its 'GenericLinkedList' implementation used in class (see the source code for lecture 5) provide an insertion method that always adds a new node to the back of the list. Assuming that the data item that is stored in the 'GenericNode' object implements the 'Comparable' interface (i.e. has standard definitions of 'compareTo(...)'), write the method
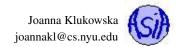
' void orderedInsert(T item)'

that adds a new node to the list in a sorted order (from smallest to largest).

Problem 4

Answer the following true/false questions. Explain your answers.

- A stack is a first in, first out structure.
- A queue is a last in, first out structure.
- A Java interface must have at least one defined constructor.
- A class that implements an interface has to implement ALL methods listed in the interface.
- In a non-empty stack, the item that has been in the stack the longest is at the bottom of the stack.
- A recursive solution to a problem is always better than an iterative solution.
- A general case of a valid recursive algorithm must eventually reduce to a base case.
- Recursive methods should be used whenever execution speed is critical.

- If you enqueue 5 elements into an empty queue, and then perform the dequeue operation 5 times, the queue will be empty again.
- If you push 5 elements into an empty stack, and then perform the peek operation 5 times, the stack will be empty again.
- In a singly linked list based implementation of a queue, at least one of the operations that add or remove an item has to traverse all the elements in the queue.
- In a singly linked list based implementation of a stack, , at least one of the operations that add or remove an item has to traverse all the elements in the stack.

Problem 5

In a circular singly linked list, the 'next' reference of the last node points back to the first node (instead of 'null'). Write a method that counts the number of nodes in a circular singly lined list. Assume that there is a 'head' reference that points to the first node in the list.

Problem 6

A doubly linked list mantains a reference to the 'head' and to the 'tail' of the list (first and last nodes, respectively). Each node has a 'next' reference and a 'previous' reference that point to the nodes after it and before it. Write the following methods that operate on a doubly linked list. Make sure that the list is still valid after the operation is performed. Make sure that your operations can handle special cases of an empty list and one element list.

A Write a method that adds a node at the beginning of the doubly linked list

B Write a method that adds a node at the end of the doubly linked list

C Write a method that removes a node at the beginning of the doubly linked list

D Write a method that removes a node at the end of the doubly linked list

Problem 7

Write a method of a 'LinedList' class that computes and returns the number of nodes stored in the linked list. Assume that there is a data field called 'head' that contains the reference to the first node. Provide both iterative and recursive implementations.

Problem 8

Write a method of a 'LinkedList' class that computes and returns the sum of the values stored in the nodes. Assume that the node definition is as follows:

```
class Node {
  int data;
  Node next;
}
```

Problem 9

Convert the following infix expressions to postfix and to prefix.

- $a - b + c$
- $a - (b/c \times d)$
- $(a - b/c) \times d$
- $a - b/(c \times d)$
- $a - (b + c \times d)/e$
- $a - (b + c) \times d/e$
- $a - (b + c) \times (d/e)$
- $(a - b) + c \times d/e$

Problem 10

Evaluate the following postfix expressions for the following values of the variables: $a = 7$, $b = 3$, $c = 12$, $d = -5$ and $e = 1$.

- $a\ b\ c + -$
- $a\ b\ c - d * +$
- $a\ b + c - d\ e * +$

Extra Challenge 1[1]

Write a method of a linked list that performs a "partition" of the values stored in the list around a value 'x'. All nodes less than 'x' should come before all nodes greater than or equal to 'x'.

Extra Challenge 2

Implement a method to check if a linked list is a palindrome. You should pass through the list only once. You may use other data structures. You may assume that the length of the list is known.

---

[1]Not for the quiz, but maybe useful for a job interview.