CSC 212 Midterm 1 - Spring 2016

College of Computer and Information Sciences, King Saud University
Exam Duration: 90 Minutes

28/02/2016

Question 1 [30 points]

1. Complete the linked implementation of the ADT *Queue*. Write down the methods: *public int length()*, *public boolean full()*, *public void enqueue(T e)*, and *public T serve()*. Write the method *public T serveEnqueue()*, which returns the first element of the queue and puts it back at the end (assume that the queue is not empty and do not call the other class methods).

```
public class LinkedQueue<T> {
        private Node<T> head, tail;
        private int size;
        public LinkedQueue() {
                tail = head = null;
                size = 0;
        }
}
```

2. Write the method *public List<T>extract(int[] pos)*, member of the class *LinkedList*, which returns a list of the elements located at the positions specified in the input array *pos* (the numbering starts with 0 at the head). Assume that all positions are valid, unique and listed in increasing order. The current list is not changed by the call. **Do not call any methods on the current list and do not use any data structure other than the returned list.**

   Example 1.1. *Given the list* $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F \rightarrow G$, *and* $pos = \{1, 3, 4\}$, l.extract(pos) *returns the list* $B \rightarrow D \rightarrow E$. *For* $pos = \{0, 2\}$, *the returned list is* $A \rightarrow C$.

Question 2 [35 points]

For each of the two following methods, write down the S/E, frequency and total for every line, the total number of steps of the method and its O notation.

1.
```
1  int func(int n) {
2          int t = 0;
3          for (int i = 5; i <= n + 3; i++) {
4                  t += i;
```

```
5                   for (int j = n - 3; j < n - 2; j++) {
6                           t += j;
7                   }
8           }
9           return t;
10  }
```

2.

```
1   void func2(int n) {
2           int k = n;
3           for(int i = 0; i < 2; i++){
4                   int j = 0;
5                   while(j <= k) {
6                           System.out.print(j);
7                           j++;
8                   }
9           }
10  }
```

Question 3 [35 points]

1. Write the method *direction*, member of the class *DoubleLinkedList*, which accepts an element e and returns the direction of e relative to the current pointer. It returns -1 if e is left of current, 0 if e is exactly on current, and 1 if e is right of current. Assume that e exists in the list. The method signature is *public int direction(T e)*. **Do not call any methods of the class *DoubleLinkedList* and do not use any other data structure.**

   Example 3.1. *Assume the list is: $A, B, C, D, E, \mathbf{F}, G, H$ and current is on F. Calling direction("B") returns -1. Calling direction("F") returns 0. Calling direction("G") returns 1.*

2. Write the method *inBetween*, user of the ADT *List*, that receives a list l and two elements $e1$ and $e2$ and returns the number of elements between $e1$ and $e2$. Assume that both $e1$ and $e2$ exist in the list l, $e1$ appears before $e2$ and there are no duplicates. The signature for the method is *public int inBetween(List<T>l, T e1, T e2)*. **Do not use any other data structures.**

   Example 3.2. *If $l : A, B, C, D$ and the elements $e1$ and $e2$ are A and D respectively, then the method should return 2. If $l : A, B, C$ and $e1$ and $e2$ are A and B. then the method should return 0.*

Specification of ADT List

- findFirst ( ): **requires**: list L is not empty. **input**: none. **results**: first element set as the current element. **output**: none.
- findNext ( ): **requires**: list L is not empty. Current is not last. **input**: none. **results**: element following the current element is made current. **output**: none.

- retrieve (Type e): **requires**: list L is not empty. **input**: none. **results**: current element is copied into e. **output**: element e.
- update (Type e): **requires**: list L is not empty. **input**: e. **results**: the element e is copied into the current node. **output**: none.
- insert (Type e): **requires**: list L is not full. **input**: e. **results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.
- remove ( ): **requires**: list L is not empty. **input**: none. **results**: the current element is removed. If L is empty, current will point to null. If the next element exists, it is made current, else the first element is made current. **output**: none.
- full (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L has reached the maximum then flag is set to true otherwise false. **output**: flag.
- empty (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **output**: flag.
- last (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the last element is the current element then flag is set to true otherwise false. **output**: flag.

Specification of ADT Queue

- enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.