Department of Computer Science and Software Engineering
Concordia University
COMP 352: Data Structure and Algorithms
Fall 2015
Assignment 3
Due date and time: Friday November 6th, 2015 by midnight

Written Questions (30 marks):

Q.1

Draw a single binary tree that gave the following traversals:

Inorder: S A E U Y Q R P D F K L M

Preorder: F A S Q Y E U P R D K L M

Q.2

Assume that the binary tree from Question 1 above is stored in an array-list as a complete binary tree as discussed in class. Specify the contents of such an array-list for this tree.

Q.3

Give an algorithm for computing the depths of all the nodes of a tree T, where *n* is the number of nodes of *T*, in O (n)-time.

Programming Question (70 marks):

In this programming assignment, you will design and implement your own version of *Tree* ADT and the *Inorder* traversal algorithm. You are not allowed to use any of the built-in classes or interfaces provided by Java, which provide similar operations. To simplify your task we assume your Tree holds an arithmetic expression (you can refer to Assignment 2 for the different possible expressions).

Your implementation of the *Tree* ADT must include at least all the methods indicated below. In addition, this class must use a linked list. This linked list, its implementation and manipulations, should never be seen by the user of the *Tree* ADT.

You must implement the following methods:

getElement( ):	returns the element stored at this position
root():	returns the position of the root of the tree
parent(p):	returns the position of the parent of position p (or null if p is the root)
children(p):	returns an iterable collection containing the children of position p.
numChildren(p):	returns the number of children of position p.
left(p):	returns the position of the left child of p
right(p):	returns the position of the right child of p
siblings(p):	returns the position of the siblings of p
isInternal(p):	returns true if position p has at least one child
isExternal(p):	returns true if position p does not have any children

isRoot(p):	returns true if position p is the root of the tree
size( ):	returns the number of positions that are contained in the tree
***isEmpty( )*:**	returns true if the tree does not contain any positions
height( ):	recursive method that returns the height of a binary tree
numleaf( ):	recursive method that returns a count of the number of leaf nodes in a binary tree
clone( ):	recursive method that returns a clone (complete copy) a binary tree.

In addition, the following methods are required for the *Inorder* traversal algorithm:

Iterator( ):	returns an iterator for all elements in the tree
Positions( ):	returns an iterable collection of all positions in the tree

1. Write the pseudo code description of your *Tree* ADT, the *Inorder* traversal, and the recursive *clone* method.
2. Implement a well formatted and documents java code for your *Tree* and the *Inorder* traversal (provide all the required classes).

3. Implement a driver (main class) that tests your implementation and functionality of your *Tree* and the *Inorder* traversal. Your driver must have sufficient test cases to validate your entire implementation. You can use the test logs (from assignment 2) for the different and sufficiently complex arithmetic expressions that use all types of operators (including parentheses) in varying combinations.

4. How does the complexity change if the underlying structure is an array instead of a linked list?

5. Briefly explain the time and memory complexity for both underlying implementation: linked list and array. Discuss the merits of each of the two implementations.

<u>Both</u> the written part and the programming part must be done <u>individually</u> (no groups are permitted). Submit all your answers to written questions in PDF (no scans of handwriting) or text formats only. Please be concise and brief (less than ¼ of a page for each question) in your answers. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into one .zip or .tar.gz file and submitted via EAS. You may upload at most one file to EAS.

For the programming component, you must make sure that you upload the assignment to the correct directory of Assignment 3 using EAS. Assignments uploaded to the wrong directory will be discarded and no resubmission will be allowed.