PROGRAMMING II – ABSTRACT DATA TYPES

Unassessed Coursework 3 : *Trees* and *Binary Search Trees*.

The aims of this coursework are to practice with:

- implementation aspects of binary trees, general trees and binary search tree;
- use of these ADTs to solve small problems.

1) Given the access procedures of a binary tree given in slides 12 and 13 of Unit 4, what tree does the following sequence of statements produce?

```
BinaryTree<Integer> t1 = new BinaryTree<Integer>(new Integer(2));
t1.attachLeft(new Integer(5));
t1.attachRight(new Integer(3));
t1 = new BinaryTree<Integer>(new Integer(10), t1, null);
BinaryTree<Integer> t2 = new BinaryTree<Integer>(new Integer(8));
t2.attacheLeft(new Integer(6));
t2. attachRight(new Integer(7));
BinaryTree<Integer> s = new BinaryTree<Integer>(new Integer(9), t1, t2);
```

2) Consider the following three access procedures of a `BinaryTree<T>` interface that display, respectively, the elements in a binary tree according to the pre-order, in-order and post-order traversal algorithms. Give an iterative implementation of each of these three access procedure as part of the class `LinkedBasedBinaryTree<T>` that implements the interface `BinaryTree<T>`. You can assume the type `T` of the elements in the tree to have a `toString` method.

```
public void preOrderDisplay();
//post: displays the elements of a binary tree in a pre-order ordering

public void inOrderDisplay();
//post: displays the elements of a binary tree in an in-order ordering

public void postOrderDisplay();
//post: displays the elements of a binary tree in a post-order ordering
```

3) Consider the following access procedure of `BinaryTree<T>` interface that displays the elements in a binary tree per level (i.e. breadth first traversal). Give an iterative implementation of this access procedure as part of the class `LinkedBasedBinaryTree<T>` that implements the interface `BinaryTree<T>`. You can assume the type `T` of the elements in the tree to have a `toString` method.

```
public void levelOrderDisplay();
//post: displays the elements of a binary tree level per level.
```

4) Consider the following method of a client program that takes in input a binary tree of integers and returns the maximum integer in the tree. Provide also the implementation of any additional access procedures of the interface BinaryTree<K> given in Unit 4 that you might found useful.

```
public int maxElement(BinaryTree<Integer> myTree);
//post: Returns the maximum integer element in myTree if not empty.
//post: Otherwise it returns the Java minimum value of an integer.
```

5) Consider the following method of a client program that searches in a given Binary Tree of Strings a given String, and returns a binary tree rooted at the node where the element is found. Give a recursive implementation of this method.

```
public BinaryTree<String> find(BinaryTree<String> myTree,
                                                   String elem)
```

//post: Returns the binary tree rooted at the node where `elem` is found.
//post: Returns an empty tree if no such `elem` exists.

6) Assume the availability of an ADT *`List`*`<Integers>` and an ADT *`BST<Integer, String>`* with its own access procedures, as defined in Slide 5 of Unit 5, including also access procedures `BST<Integer, String> getLeftSubTree()` and `BST<Integer, String>` *`getRightSubTree`*. Implement the following high-level procedures, using recursion:

```
public List<Integer> TreetoList(BST<Integer, String> myTree).
//post: Returns a list containing the keys values in the tree in descending order.

public List<Integer> getOddKeys(BST<Integer, String> myTree)
//post: Returns a list containing the odd (i.e. not divisible by 2) key values in
//post: the tree in ascending order.
```

7) Consider the following method of a client program that takes in input a Binary Search Tree of Integers and a range of integers defined by the given low and high parameters. The method prints all the elements in the tree with key value in the given range. Give an implementation of this method.

```
public void rangeQuery(BST<Integer> myTree, int low, int  high)
```

//post: Prints the nodes in `myTree` with key within the range between `low` and `high` included.