CSE 505 Fall 2015
Assignment # 3 – Functional Programming

Assigned: October 22, 2015
Due: November 9, 2015 (11:59 pm)

Note: This assignment may be done by a pair of students.

Problem 1. Assume the following ML type for binary trees:

```
datatype 'a tree = leaf | node of 'a * 'a tree * 'a tree;
```

Complete the definition below for a function `insert(i, tr)` which will insert a value `i` into `tr` (if `i` is not present in `tr`) so as to maintain the *binary search tree* property.

```
fun insert(i, leaf) =  __________________
  | insert(i, node(v,left,right)) =  ___________________;
```

In the above code, ML will assume `v` to be of type `int` by default. In completing the definition of `insert`, note that you do not update the input tree; rather, you need to construct a new tree in which the value to be inserted is placed in the correct position. Test your definition by running the following function, making sure to place the code for `testcase` *after* `insert`:

```
fun testcase() =
        let val t1 = node(100,leaf,leaf);
            val t2 = insert(50, t1);
            val t3 = insert(150, t2);
            val t4 = insert(200, t3);
            val t5 = insert(125, t4);
            val t6 = insert(175, t5);
            val t7 = insert(250, t6);
            val t8 = insert(25, t7);
            val root = insert(75, t8)
         in root
        end;
```

Prepare a file **insert.sml** containing the type definition for `tree` as well as the code for `testcase and insert.`

Problem 2. Consider the following depth-first ("in order") traversal of a BST.

```
fun dfirst(leaf) = []
  | dfirst(node(v,t1,t2)) = dfirst(t1) @ [v] @ dfirst(t2);
```

Develop a tail-recursive equivalent of `dfirst`, called `dfirst2`. Develop the definition of `dfirst2` by writing a tail-recursive helper function, say `df`, which will use an *accumulator-passing style* in order to construct the answer. Nest the definition of `df` inside `dfirst2` using a `let-in-end` block. Test out `dfirst2` by running the following function:

```
fun test_dfirst2() = dfirst2(testcase());
```

Prepare a file **dfirst.sml** containing the type definition for `tree` and the code for `insert`, `testcase`, and `dfirst2`.

Problem 3. Consider the following ML definitions of the familiar higher-order functions `map` and `reduce`:

```
fun map(f, [ ])  = [ ]
  | map(f, x::t) = f(x) :: map(f, t);

fun reduce(f, b, [ ]) = b
  | reduce(f, b, x::t) = f(x, reduce(f, b, t));
```

An *n-ary tree* is a generalization of a binary tree in which each internal node has a list of zero of more subtrees, and each leaf node holds a value. Below is an ML type definition for an n-ary tree:

```
     datatype 'a ntree = leaf of 'a  |  node of  'a ntree list;
```

a. Using `map`, define a function `subst(tr,v1,v2)` which returns a new `ntree` in which all occurrences of value `v1` in `tr` are replaced by `v2` in the output tree. For example,

```
subst(node([leaf("x"), node([leaf("y"), leaf("x"), leaf("z")])]),
     "x", "w") =
      node([leaf("w"), node([leaf("y"), leaf("w"), leaf("z")])])
```

b. Using `reduce`, define a function `toString(tr)` which concatenates all strings at the leaf nodes of a `string ntree`, adding a space between each value, and returns the resulting string. For example,

```
toString(node([leaf("x"),node([leaf("y"),leaf("x"),leaf("z")])])) =
"x y x z"
```

Note: The functions `subst` and `toString` will be recursive, but the recursive calls should be initiated from `map` and `reduce` respectively, i.e., through the parameter `f` of `map` and `reduce`.

Prepare a file **mapreduce.sml** containing the type definitions for `ntree`, `map`, `reduce`, `subst`, and `toString`.

What to Submit:

Prepare a top-level directory named *A3_UBITId1_UBITId2* if the assignment is done by two students; otherwise, name it as *A3_UBITId* if the assignment is done solo. (Order the *UBITId*'s in alphabetic order, in the former case.) In this directory, place `insert.sml`, `dfirst.sml`, and `mapreduce.sml`. Compress the directory and submit the resulting compressed file using the `submit_cse505` command. For more details regarding online submission, see `Resources` → `Homeworks` → `Online_Submission_2015.pdf`.

End of Assignment #3