Computer Engineering 171
Homework 3: Functional Programming

Due: November 11th 9:00 am

In this assignment, you will write programs in ML and Scheme. Use the Linux machines in the Engineering Computing Center for this project. The ML interpreter is `sml` (use "`setup smlnj`" first) and the Scheme interpreter is `guile`.

1 Quicksort

On Camino you will find an implementation in ML of ***quicksort***, the sorting algorithm used previously. Translate the program to Scheme, keeping the same value bindings, function names, and parameters. Call this program `sort.scm` and submit it using Camino.

Goal: To introduce functional programming in strongly-typed and dynamically-typed languages.

Hints: You will need to use `let*` in Scheme to achieve the same effect as `let` in ML.

2 Binary Search Trees

As in the first assignment, a ***binary search tree*** is either empty, or it consists of a node with two binary search trees as subtrees. Each node holds an integer. The elements in a binary search tree are arranged so that smaller elements appear in the left subtree of a node and larger elements appear in the right subtree. In ***both*** ML and Scheme, write a program to implement binary search trees. Call your programs `tree.sml` and `tree.scm`, respectively, and submit them using Camino. Each program should have at least the following two functions:

- `insert`: given a binary search tree and an integer, inserts the integer into the tree, and returns the new tree; if the integer has already been inserted, then the tree is unchanged and the tree itself is returned
- `member`: given a binary search tree and an integer, returns true if the integer is found in the tree, and returns false otherwise (in Scheme, call this function `member?` to avoid clashing with the built-in function of the same name)

In this assignment, you ***may not*** use assignment or iteration (i.e., any Scheme function whose name ends in an exclamation point such as `set!` or the ML assignment operator). Consequently, the `insert` function must return a tree. For the Scheme program, you will have to use lists to represent the tree, as Scheme does not support user-defined datatypes. However, since ML does support user-defined datatypes, you should define your own datatype to represent the tree.

Goal: To represent structures in languages with structured types and without structured types.

Hints: For the Scheme program, I found it easiest to represent a node as a list with three elements: the data, the left subtree, and the right subtree. Both the left and right subtrees would themselves be lists with three elements. The empty list is used to represent an empty subtree.

3 Coding Guidelines

- Use cases in the ML solution for both the `insert` and `member` functions.
- Use currying in your ML solution: `insert` should have type `tree` → `int` → `tree`, and `member` should have type `tree` → `int` → `bool`. If your tree type is polymorphic, which is encouraged but not required, then your functions would use `int tree` rather than just `tree`.
- You do not need to write extra functions such as `inorder` or `preorder` but it is okay to include them in your solution.
- For the Scheme solution to `quicksort`, you are explicitly asked to keep the same binding, function, and parameter names. Include bindings for the head and tail of the list parameter. For the Scheme solution to the binary search tree, you are free to use `let` or `let*` to introduce additional bindings if you like, and to write additional functions such as `left-child`, `right-child`, etc. if you wish.