CSE505 – Fall 2012
Assignment 3: Functional Programming and ML
Assigned Thurs, Oct 11
Due Mon, Oct 22
Note: The midterm exam will be held in class on **Weds, Oct 24**

2 [40%] Consider the following type definition for a binary-tree dictionary of key-value pairs, where the key is a string and the value is an integer:

datatype treedict = leaf **of** string * int | node **of** treedict * treedict

Write an ML function lookup: string*treedict → string that behaves as follows: Suppose a dictionary, dict, has the value 100 for "apple", then lookup("apple", dict) will return the string "the value of apple is 100". And suppose "orange" is not in the dictionary, then lookup("orange", dict) will return the string "orange not found".
When recursively searching a large tree-structured dictionary, it would be efficient to raise an exception, say 'found **of** string*int', when the search is successful. Similarly, raise an exception, say 'notfound **of** string', when the search is unsuccessful. Develop your solution using this approach, and construct the desired output strings in the respective handlers for these exceptions.

Submit a file, lookup.sml, containing your ML code using the online code submission tool. See the Announcements page of the course website for instructions on how to use this tool. Resources for ML may be found via the Online References page of the course website.

2 [30%] Consider the **datatype** 'a gametree = node **of** 'a * 'a gamtree **list** discussed in class. Assuming that the strength assessment function for a game position is of the form

fun assess = minimax ∘ treemap(strength) ∘ prune(5) ∘ game

where

assess: position → int
minimax: int gametree → int
strength: position gametree → int
treemap: (position gametree → int) → position gametree → int gametree
prune: int → position gametree → position gametree
game: position → position gametree

Note that 'position' is the type for a typical position (or configuration) of a game. Define treemap and prune using ML notation. Refer to the paper, "Why Functional Programming Matters," by John Hughes, available from the web, for guidance on how to write your solution.

3 [30%] Consider the following definition in a lazy functional language:

```
fun primes = sieveall(numsfrom(2));
fun sieveall(p::t) = p :: sieveall(sieve(t,p));
fun sieve(h::t, p) = if h mod p = 0 then sieve(t,p) else h :: sieve(t,p);
fun numsfrom(n) = n :: numsfrom(n+1);
```

Translate the above code into a non-lazy functional language with higher-order functions. The translation should use the approach that was illustrated in class for numsfrom.

End of Assignment 3