211hw3

Homework 3 (Due Friday 2/5/2010 at 10:00am)

Submit your `.ss` file via OWL-Space. You will need to use the "Intermediate Student" language to do Problem 18.1.15. If you want to use explicit `lambda` notation (anywhere the right hand side of a `define` statement), you will need to use the "Intermediate Student with lambda" language. You may use either intermediate level language for the entire assignment if you choose.

Required problems:

1. 14.2.4 [20 pts.]
   Note: Be sure to compare list searching with tree searching, as the problem states.
2. 16.3.3 [20 pts.]
   Notes:
   a. Test every function thoroughly (5+ examples).
   b. Be sure to include definitions for *both* variations of `du-dir`. The final sentence should read "storing a file **or** a directory in a dir structure costs 1 storage unit." In other words, given a dir structure, each directory entry (a file or a directory) contained therein costs 1 unit of storage for the bookkeeping data. For a file, this bookkeeping overhead is in addition to the size of its data.
3. 17.1.2 [20 pts.]
4. 17.6.1 [20 pts.]
   Do the problem as specified in the book.
   Extra Credit [10 pts.]: This problem can be solved more elegantly than the solution implied in the book. For the extra credit solution *ignore* the book's guidance on "writing functions that consume two complex inputs" in 17.5 and follow the guidance given in class on how to write a function that processes multiple inputs. Select *one* input as primary (the choice may be *arbitrary* in some cases). If you need to deconstruct a second argument, do it in a *auxiliary* function. Use only *one* design template in each function. Hint for solving this problem: only your auxiliary function, which has a contract and purpose statement almost identical to `merge`, should be recursive (call itself directly or indirectly) and it may need to deviate slightly from the structural recursion template. The top level `merge` function is *not* recursive.
   Note If you do the extra credit version of this problem, you do not need to write a solution as specified in the book.
5. 17.7.1 [10 pts.]
   Note: Make sure you understand section 14.4 before working on this problem. Use this data definition as a starting point:

   ```
   ; expression
   ; An expression is one of:
   ; - a number
   ; - a symbol
   ; - (make-mul e1 e2) where e1 and e2 are expressions
   ; - (make-add e1 e2) where e1 and e2 are expressions
   ```

   You are required to extend this definition to include applications, which are expressions like

   ```
   (* (f (+ 15 x)) (g x))
   ```

   Be sure to include a function template with your solution.
6. 18.1.5, parts 1, 4, & 5 [5 pts.]
7. 18.1.15 [5 pts.]

Optional problem for extra credit: [50 pts]
The fibonacci function fib is defined by the following rules (in Scheme notation):

```
(fib 0) = 0    ;; formerly (fib 0) = 1; revised to match the prevailing mathematical conventions for defining
fib
(fib 1) = 1
(fib (+ n 1)) = (+ (fib n) (fib (- n 1)))
```

A naive program for computing fib (lifted directly from the definition) runs in exponential time, i.e. the running time for `(fib n)` is proportional to `K*b**n` for some constants `K` and `b`). It is easy to write a program that computes `(fib n)` in time proportional to `n`. Your challenge is to write a program that computes `(fib n)` in *log* time assuming that all multiplications and additions take constant time, which is unrealistic for large `n`. More precisely, your program should compute `(fib n)` using only `O(log n)` addition and multiplication operations (less than `K * log n` operations for some constant `K`).
Hints: assume `n = 2**m`. Derive a recurrence for `fib 2**(m + 1)` in terms of `fib 2**m` and `fib 2**m - 1`. Initially write a program that works when n is a power of 2. Then refine it to a program that works for all n.