Assignment 3: Higher-Order Functions CS351—Fall 2008

Due 23:59 Sun 19-Oct-2008. Email *one text file* containing *all* your solutions to: `barak+cs351-hw3@cs.nuim.ie`.

1. Define `map-leaves` which takes a function and an s-expression and returns the result of applying the given function to every non-list datum inside the given s-expression.

   `(map-leaves - '((1 2) -3 -4 (((5) (())))))` ⇒ `((-1 -2) 3 4 (((-5) (()))))`

   `(map-leaves list '(a (b c) d))` ⇒ `((a) ((b) (c)) (d))`

   Solution:

   ```
   (define map-leaves
     (lambda (f s)
       (cond ((pair? s) (cons (map-leaves f (car s))
                              (map-leaves f (cdr s))))
             ((null? s) s)
             (else (f s)))))
   ```

   or

   ```
   (define map-leaves
     (lambda (f s)
       (map-leaves-2 (lambda (x1 x2) (f x1))
                     s s)))
   ```

2. Define `flip` which has the following behaviour:

   `((flip /) 2 10)` ⇒ 5

   `((flip list) 'aye 'bee)` ⇒ `(bee aye)`

   `((flip append) '(a b c) '(1 2 3))` ⇒ `(1 2 3 a b c)`

   Solution:

   ```
   (define flip
     (lambda (f)
       (lambda (y x)
         (f x y))))
   ```

3. Define `map-leaves-2` which takes a binary function and two s-expressions, and applies to the given function to structurally corresponding elements of the two s-expressions in which one of the two elements is not a list. *E.g.*,

   `(map-leaves-2 + '(1 2 (3 4)) '(10 20 (30 40)))` ⇒ `(11 22 (33 44))`

   `(map-leaves-2 list '(1 2 (3 4)) '(10 20 (30 40)))`
   ⇒ `((1 10) (2 20) ((3 30) (4 40)))`

   `(map-leaves-2 cons '(1 2 (3 4)) '(() (a b) ((c d e) (f))))`
   ⇒ `((1) (2 a b) ((3 c d e) (4 f)))`

Solution:

```
(define map-leaves-2
  (lambda (f s1 s2)
    (cond ((and (pair? s1) (pair? s2))
           (cons (map-leaves-2 f (car s1) (car s2))
                 (map-leaves-2 f (cdr s1) (cdr s2))))
          ((or (null? s1) (null? s2)) ’())
          (else (f s1 s2)))))
```

4. Define `swizzle-leaves` which takes an s-expression and an association list and switches each item in the s-expression which appears as a key in the alist for the corresponding associated item.

   `(swizzle-leaves ’(a (b c a) d) ’((a aye) (c sea)))` ⇒ `(aye (b sea aye) d)`

Solution:

```
(define swizzle-leaves
  (lambda (s alist)
    (cond ((null? s) s)
          ((pair? s) (cons (swizzle-leaves (car s) alist)
                           (swizzle-leaves (cdr s) alist)))
          (else ((maybe-lookup-curried alist) key)))))

(define maybe-lookup-curried
  (lambda (alist)
    (lambda (key)
      (cond ((assoc key alist) => cadr)
            (else key)))))
```

or

```
(define swizzle-leaves
  (lambda (s alist)
    (map-leaves (maybe-lookup-curried alist) s)))
```

or

```
(define swizzle-leaves
  (lambda (s alist)
    (map-leaves (lambda (x)
                  (cond ((assoc x alist) => cadr)
                        (else x)))
                s)))
```

5. Consider the following “little language” of constrained binary numeric expressions:

$$\begin{aligned} \langle \mathit{expr} \rangle &:= \langle \mathit{number} \rangle \mid \textbf{(}\ \langle \mathit{expr} \rangle\ \langle \mathit{op} \rangle\ \langle \mathit{expr} \rangle\ \textbf{)} \\ \langle \mathit{op} \rangle &:= \textbf{+} \mid \textbf{*} \mid \textbf{-} \mid \textbf{/} \end{aligned}$$

where ⟨*number*⟩ is a native Scheme number. (Note that all those parenthesis are mandatory.) Define `eval-expr` which evaluates such an *expr*, represented in the obvious way as a Scheme s-expression, using the obvious semantics.

`(eval-expr 17)` ⇒ `17`

`(eval-expr '(17 + 1))` ⇒ `18`

`(eval-expr '(17 + (10 / 2)))` ⇒ `19`

`(eval-expr '(0 - (((1 + (1 + 1)) + 1) / 2)))` ⇒ `-2`

Solution:

```
(define eval-expr
  (lambda (e)
    (cond ((pair? e)
           ((lookup-op (cadr e))
            (eval-expr (car e))
            (eval-expr (caddr e))))
          (else e))))

(define lookup-op
  (lambda (op)
    (cond ((equal? op '+) +)
          ((equal? op '-) -)
          ((equal? op '*) *)
          ((equal? op '/) /))))
```

6. *(Optional)* If you encountered any problems with the assignment, or have any comments on it, or other comments or suggestions, I would appreciate hearing them. As practice for actual work, where weekly reports are not unusual, please embody these in a brief report.

Solution:

This is the best class ever. My only suggestion: longer harder assignments. And more of them!

Honor Code: You may discuss these with others, but please write your answers by yourself and without reference to communal notes. In other words, your answers should be *from your own head.*