

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))
```

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)
```

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)))
```

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)
```

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

$$\langle expr \rangle := \langle number \rangle \mid (\langle expr \rangle \langle op \rangle \langle expr \rangle)$$
$$\langle op \rangle := + \mid * \mid - \mid /$$

where $\langle number \rangle$ 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
```

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.

<p>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 <i>from your own head</i>.</p>
