COEN 166/266 - Spring 2015

Homework - Exploring Scheme

Description

For this assignment, implement the following procedures in scheme, using **only** the constructs described in the "Scheme Quick Reference" document. These exercises are intended to get you comfortable with the Scheme development environment, the language constructs, list-based data types, and the use of recursion as an iteration construct.

`swap-adjacent`

Takes a list as a parameter and returns a list with the same elements but adjacent elements swapped. For example:

```
guile> (swap-adjacent '(1 2 3 4 5))
(2 1 4 3 5)
guile> (swap-adjacent '(1 2))
(2 1)
guile> (swap-adjacent '())
()
guile> (swap-adjacent '((1 2) 3 (4) () 5 6 (7)))
(3 (1 2) () (4) 6 5 (7))
```

`rotate-left`

Takes two parameters: a value indicating the number of positions to rotate and a list to rotate. Returns the input list with all values rotated left by the number of positions specified. For example:

```
guile> (rotate-left 1 '(a b c))
(b c a)
guile> (rotate-left 3 '(a b c))
(a b c)
guile> (rotate-left 0 '(a b c))
(a b c)
guile> (rotate-left 100 '(a b c))
(b c a)
```

`palindrome?`

Takes a list as a parameter and returns a boolean result indicating whether that list is the same forward and backwards. For example:

```
guile> (palindrome? '(a b c))
#f
guile> (palindrome? '(a b a))
#t
guile> (palindrome? '((a) b c (d e) c b (a)))
#t
guile> (palindrome? '())
#t
```

Submission

Submit (via Camino) a copy of a single .scm file implementing all of these procedures, plus any helper procedures you choose to implement. Please be sure to use the names and parameter syntax specified in the description, to facilitate automated testing.

Due Date

13 April, 2015