Scheme Tutorial Exercises

Fall 2003

Problem Set 3: Basic Higher-order functions

21-25. Rewrite the functions in exercises 11-15 using *map*, *filter*, *foldl*, or *foldr*.

26. Define the function *compose-func*, which consumes two functions of one argument, and returns the composition of these functions. For example:

    ((*compose-func first rest*) '(a b c d))

    > b

27. Define the function *flatten*. It consumes a list of sublists of numbers, and produces a list of all numbers in the sublists. For example:

    (*flatten* '((1 2) (3 4 5) (6)))

    > '(1 2 3 4 5 6)

    Write two version of the function: one that uses *foldr* and one that doesn't.

28. Use *foldr* to define the function *bucket*. It consumes a list of numbers, and returns a list of sublists of adjacent equal numbers. For example:

    (*bucket* '(1 1 2 2 2 3 1 1 1 2 3 3))

    > '((1 1) (2 2 2) (3) (1 1 1) (2) (3 3))

29. Define the function *tree-map*. It consumes a function *f* over strings and a family-tree *t* (See exercise 17), and produces a tree where *f* has been applied to each name in *t*.

30. Use *tree-map* to define *add-last-name*. This function consumes a family tree and a string, and produces a tree where the string has been appended to each name.

    Hint: The Scheme function *string-append* takes two strings and returns a new string representing their concatenation.