CS3733-B00: Software Engineering

MzScheme Programming Exercises

These exercises are meant to give you practice in data modeling and MzScheme programming. Most of the problems should be similar to ones you have written in previous programming classes or experience. **If any of the problems other than the last one are beyond your prior programming experience (in any language), you might not be ready to take this class!**

Note that *almost none of these programs require assignment operators*, such as **set!** or **set-car!**. **Your goal is to write these programs with as few assignment operators as possible!** You should be able to do all but problem 6b without assignment operators.

1. Program *duple*, which consumes a number n and an item x and produces a list consisting of n copies of x.

;; Examples
> (*duple* 2 3)
(3 3)
> (*duple* 4 '(ho ho))
((*ho ho*) (*ho ho*) (*ho ho*) (*ho ho*))

2. Program *invert*, which consumes a list of two-element lists and produces a list with each two-element list reversed.

;; Examples
> (*invert* '((a 1) (a 2) (b 1) (b 2)))
((1 *a*) (2 *a*) (1 *b*) (2 *b*))

3. Program *swapper* which consumes two items and a list and returns the same list as the original but with all occurrences of the first item replaced with the second and vice-versa.

;; Examples
> (*swapper* 'a 'd '(a b c d))
(*d b c a*)
> (*swapper* 'a 'd '(a d () c d))
(*d a* () *c a*)
> (*swapper* 'x 'y '((x) y (z (x))))
((*y*) *x* (*z* (*y*)))

4. Program *mergesort*, which consumes two lists of numbers and returns the list sorted in ascending order. Use a helper function *merge* which consumes two lists sorted in ascending order and returns a sorted list of all numbers in the two input lists.

;; Examples
> (*merge* '(1 4) '(1 2 8))
(1 1 2 4 8)
> (*merge* '(35 62 81 90 91) '(3 83 85 90))
(3 35 62 81 83 85 90 90 91)
> (*mergesort* '(8 2 5 2 3))
(2 2 3 5 8)

5. A set of shapes contains circles (defined by center point and radius), rectangles (defined by top-left corner, width, and height), and squares (defined by top-left corner and width).

   Write the following programs over shapes:

(a) *area*, which consumes a shape and returns its area

(b) *translate*, which consumes a shape and a number (the x-offset) and returns a shape with the same dimensions as the original, but moved horizontally by the x-offset.

6. A gradebook contains certain information for each student: name, id number, and a list of numeric scores on homework assignments.

   Write the following programs over gradebooks:

   (a) *total-points*, which consumes a student name and a gradebook and returns the total homework points earned by the student so far.

   (b) *add-grade*, which consumes a student name, a homework grade, and a gradebook and adds the grade to the gradebook for the named student.

   Without editing any of your previous code for this problem, add a class year (freshman, etc) for each student and write a program *grade-warning* which consumes a gradebook and a class year and produces a list of names of all students in that year who have fewer than 50 homework points.

7. A binary tree is a tree in which each node contains a number and accessors to (up to) two other nodes (left and right). A binary search tree is a binary tree with the following property: at each node of the tree, the number at that node must be larger than or equal to the numbers in all nodes accessible down the left branch and smaller than or equal to all nodes accessible down the right branch.

   Write the following programs over binary (search) trees:

   (a) *bst?*, which consumes a binary tree and returns a boolean indicating whether the tree is a binary search tree.

   (b) *bst-order*, which consumes a binary search tree and returns an ordered list of all numbers stored in the tree.

   (c) *bst-add*, which consumes a binary search tree and a number and returns a binary search tree that contains the new number and all of the numbers that were in the original tree.

8. A file system consists of files organized into potentially nested subdirectories. For this problem, we are interested in storing the name, size, and contents of each file, and the name and contents (files and subdirectories) of each directory.

   Write the following programs over filesystems:

   (a) *total-size*, which consumes a filesystem and returns the total size of all files in the filesystem.

   (b) *contains-file?*, which consumes a filesystem and a file name and returns a boolean indicating whether the filesystem contains the given file.