Problem Set 1
Due September 11/September 18

A. Paper Problems - Due September 11

Problem 1 (PostFix + Pairs)
Do Exercise 3.37 on page 96 of the text.

Problem 2 (PostText)
Do Exercise 3.44 on page 99 of the text.

B. Programming Problems - Due September 18

In the next two problems, you will extend the PostFix interpreter described in Section 1.2.2 of the Scheme supplement to implement the extensions described in the first two problems. You may want to read appendix A of the Scheme supplement as early as possible.

We especially urge people to work in groups on programming assignments. Remember that you must write up your own solutions and attribute any joint work. See the collaboration policy described in Handout 3 for more information.

First, you should play with the existing PostFix interpreter. Download ps1/postfix.scm from the course directory and start the interpreter by evaluating the following Scheme+ expressions:

(load "postfix.scm")
(pf-den-repl)

This will start a read-eval-print loop. (There is nothing to turn in for this part—just familiarize yourself with the interpreter.)

Problem 3 (PostFix + Pairs + \{dup\} Lab)

Extend the interpreter to handle the pair, left, and right commands of Problem 1 as well as the dup command discussed in Section 3.8 of the text.

As mentioned in Handout 3 (General Information), your solutions for this and all programming problems should clearly indicate what modifications you have made and include an English explanation of what those modifications do. Demonstrate that your program correctly solves the problem by testing it on a sufficient test suite.
Problem 4 (PostText Lab)

Extend the PostFix interpreter to implement the extensions described in Problem 2.

a. Implement a dictionary abstraction by writing Scheme+ procedures that have the following interface:

- (dict-empty) creates an empty dictionary.
- (dict-bind name value dict) returns a new dictionary that extends dict with a binding between name and value.
- (dict-lookup name dict) returns the value associated with name in the given dictionary, dict.

Based on the suggestion in Exercise 3.44(b), you should represent dictionaries as one-argument procedures that map symbols to values.

b. Define a new state datatype as follows:

   (define-datatype state (make-state stack dict))

Modify the evaluation procedures in the interpreter so that they map states to states rather than mapping stacks to stacks.

c. Extend the interpreter to handle the I, def, and ref commands described in Exercise 3.44. One of your test cases should be a non-terminating program.