```
;; This is a denotational interpreter for PostFix
;; Built on Wed Sep  7 12:45:14 EDT 1994
;; Includes:
;;    pf-syntax.scm
;;    repl.scm
;;    pf-den-interp.scm

;;-------------------------------------------------
;; pf-syntax.scm
;;;-----------------------------------------------------------------------------
;;; PF-SYNTAX.SCM
;;;
;;; Postfix syntax and parsing
;;;-----------------------------------------------------------------------------

;;;-----------------------------------------------------------------------------
;;; Datatypes

(define-datatype program
($prog (listof command)))

(define-datatype command
($int int)
($seq (listof command))
($pop)
($swap)
($dup)
($sel)
($exec)
($arithop (-> (int int) int))
($relop   (-> (int int) bool))
)

;;;-----------------------------------------------------------------------------
;;; Parsing

(define (pf-program sexp)
(match sexp
((list->sexp lst) ($prog (pf-sequence lst)))
(_ (error "Ill-formed program"))))

(define (pf-sequence lst)
(map pf-command lst))

(define (pf-command sexp)
(match sexp
```

```
39. ( (int->sexp n)      ($int n)                    )
40. ( (list->sexp lst)   ($seq (pf-sequence lst))       )
41. ( 'pop               ($pop)                      )
42. ( 'swap              ($swap)                     )
43. ( 'exec              ($exec)                     )
44. ( 'sel               ($sel)                      )
45. ( 'dup               ($dup)                      )
46. ;; Below, arithop and relop operations are functions, not symbols!
47. ( 'add               ($arithop +)                )
48. ( 'sub               ($arithop -)                )
49. ( 'mul               ($arithop *)                )
50. ( 'div               ($arithop quotient)            ) ; integer division
51. ( 'lt                ($relop   <)                )
52. ( 'eq                ($relop   =)                )
53. ( 'gt                ($relop   >)                )
54. ( _                  (error "Unrecognized command"
                    i.   sexp)                   )
55. ))
```

```
;;-------------------------------------------------
;; repl.scm
(define (make-repl evaluator prompt parser unparser)
(lambda ()
(let loop ()
(display "\n\n")
(display prompt)
(display " ")
(let ((sexp (read)))
(if (eq? sexp 'quit)
    a. (display "\nGoodbye!\n")
    b. (begin
    c. (display "\n")
    d. (display (unparser (evaluator (parser sexp))))
    e. (loop)))))))
```

```
;;-------------------------------------------------
;; pf-den-interp.scm
;;;-----------------------------------------------------------------------------
;;; PF-DEN-INTERP.SCM
;;;
;;; A PostFix interpreter based on the denotational semantics for PostFix.
;;; This is a "curried version" in which stacks are passed in a curried
;;; style. This corresponds directly to the denotational semantics.
;;; Uses Scheme's ERROR instead of error-stack to model errors.
;;;
;;; EXERCISES:
;;; * Model errors explicitly (don't forget divide-by-zero).
;;; * Write in terms of WITH-INT&STACK; WITH-ERROR
;;; * Add DUP.
;;; * Modify so that EVAL-COMMAND and EVAL-COMMANDS are take stack in
;;;   uncurried fashion.
;;;
;;;-----------------------------------------------------------------------------

;;;-----------------------------------------------------------------------------
;;; Evaluation

(define-datatype den-val
(int->den-val int)
(xform->den-val (-> (stack) stack)))

;; eval-program: (-> (program) den-val)
(define (eval-program pgm)
(match pgm
(($prog seq) (top ((eval-commands seq) (empty-stack))))
))

;; eval-commands: (-> ((listof command)) (-> (stack) stack))
(define (eval-commands seq)
(match seq
((null) identity)
((cons com coms) (o (eval-commands coms) (eval-command com)))
))

        ;; eval-command: (-> (command) (-> (stack) stack))
        (define (eval-command cmd)
        (match cmd
        ( ($int i)      (push (int->den-val i))                         )
        ( ($seq s)      (push (xform->den-val (eval-commands s)))           )
```

```
105.    ( ($pop)       pop                                    )
106.    ( ($swap)      (with-value
            i.  (lambda (v1)
            ii. (with-value
                    1. (lambda (v2)
                    2. (o (push v2) (push v1))))))              )
107.    ( ($sel)       (with-value
            i.  (lambda (else)
            ii. (with-value
                    1. (lambda (then)
                    2. (with-integer
                    3. (lambda (test)
                    4. (if (= test 0)
                            a. (push else)
                            b. (push then))))))))               )
108.    ( ($exec)      (with-transform identity)                 )
109.    ( ($arithop op) (with-integer
            i.  (lambda (i1)
            ii. (with-integer
                    1. (lambda (i2)
                    2. (push (int->den-val (op i2 i1)))))))     )
110.    ( ($relop op)  (with-integer
            i.  (lambda (i1)
            ii. (with-integer
                    1. (lambda (i2)
                    2. (push (int->den-val (if (op i2 i1) 1 0)))))))  )
111.    ))


112.    ;;;-------------------------------------------------------------------------
113.    ;;; Auxiliary Functions

114.    (define (empty-stack) '())

115.    (define (push val)
116.    (lambda (stack)
117.    (cons val stack)))

118.    (define (with-value proc)
119.    (lambda (stack)
120.    (match stack
121.    ((null) (error "Empty stack"))
122.    ((cons v s) ((proc v) s)))))

123.    (define top (with-value (lambda (top) (lambda (rest) top))))
124.    (define pop (with-value (lambda (top) (lambda (rest) rest))))
```

```
125.    (define (with-integer proc)
126.    (with-value
127.    (lambda (v)
128.    (match v
129.    ((int->den-val i) (proc i))
130.    (_ (error "Transform where integer expected"))))))

131.    (define (with-transform proc)
132.    (with-value
133.    (lambda (v)
134.    (match v
135.    ((xform->den-val t) (proc t))
136.    (_ (error "Integer where transform expected"))))))

137.    (define (identity x) x)

138.    (define (o f g)
139.    ;; Function composition
140.    (lambda (x)
141.    (f (g x))))


142.    ;;;----------------------------------------------------------------------------
143.    ;;; Top-level

144.    (define (unparse-value value)
145.    (match value
146.    ((int->den-val i) (int->sexp i))
147.    ((xform->den-val s) 'executable)
148.    ))

149.    (define pf-den-repl (make-repl eval-program
                                a. 'pf-den>
                                b. pf-program
                                c. unparse-value))
```