```
(module lang
  (lib "eopl.ss" "eopl")                ; use the eopl language level

  (let ((time-stamp "Time-stamp: <2005-01-31 22:52:12 Owner>"))
    (eopl:printf "lang3-3/lang.scm ~a~%"
      (substring time-stamp 13 29)))

;;; lang3-3/lang.scm
;;; Author: Mitch Wand
;;; Wed Dec 22 15:58:21 2004

;;; language of section 3.6 in book:  like lang3-3, but with n-ary
;;; primitive operations, let declarations, procedures, applications,
;;; and letrec declarations.

  (require "../drscheme-init.scm")

  (provide (all-defined))

  ;;;;;;;;;;;;;;;; grammatical specification ;;;;;;;;;;;;;;;;

  (define the-lexical-spec
    '((whitespace (whitespace) skip)
      (comment ("%" (arbno (not #\newline))) skip)
      (identifier
        (letter (arbno (or letter digit "_" "-" "?")))
        symbol)
      (number (digit (arbno digit)) number)
      (number ("-" digit (arbno digit)) number)
      ))

  (define the-grammar
    '((program (expression) a-program)

      (expression (number) lit-exp)

      ;; application of a primitive
      (expression
        (primitive "(" (separated-list expression ",") ")")
        primapp-exp)

      (expression
        ("if" expression "then" expression "else" expression)
        if-exp)

      (expression (identifier) var-exp)

      ;; allow many declarations in a let
      (expression
        ("let" (arbno identifier "=" expression) "in" expression)
        let-exp)

      ;; allow many bound variables in a procedure
      (expression
        ("proc" "(" (separated-list identifier ",") ")" expression)
        proc-exp)

      ;; allow many operands in an application
      (expression
        ("(" expression (arbno expression) ")")
        app-exp)

      ;; allow many declarations of n-ary procedures in a letrec
      (expression
        ("letrec"
          (arbno identifier "(" (separated-list identifier ",") ")"
            "=" expression)
          "in" expression)
        letrec-exp)
```



```
      ;; primitives
      (primitive ("+")      add-prim)
      (primitive ("-")      subtract-prim)
      (primitive ("*")      mult-prim)
      (primitive ("add1")  incr-prim)
      (primitive ("sub1")  decr-prim)
      (primitive ("zero?") zero-test-prim)

      ;; new for mutable-variables:

      (expression
        ("set" identifier "=" expression)
        varassign-exp)

      ;; new for mutable pairs:

      (primitive ("pair")      pair-prim)
      (primitive ("left")      left-prim)
      (primitive ("right")     right-prim)
      (primitive ("setleft")   setleft-prim)
      (primitive ("setright")  setright-prim)

      ))

  ;;;;;;;;;;;;;;;; sllgen boilerplate ;;;;;;;;;;;;;;;;

  (sllgen:make-define-datatypes the-lexical-spec the-grammar)

  (define show-the-datatypes
    (lambda () (sllgen:list-define-datatypes the-lexical-spec the-grammar)))

  (define scan&parse
    (sllgen:make-string-parser the-lexical-spec the-grammar))

  (define just-scan
    (sllgen:make-string-scanner the-lexical-spec the-grammar))

  )
```