```
;;;-----------------------------------------------------------------------
;;; PS2.SCM
;;;
;;; Handy procedures for 6.821 Problem Set 2, Fall '98.
;;;-----------------------------------------------------------------------


;;;-----------------------------------------------------------------------
;;; General simplifier

(define (make-language-simplifier node-handler node=?)
(lambda (rules)
(lambda (node)

(define (simplify node)
(fixed-point simplify-one-pass node))

(define (simplify-one-pass node)
(node-handler (apply-all node)
(lambda (subnodes make-node)
        (apply make-node
                (map simplify-one-pass subnodes)))))

(define (apply-all node)
(fixed-point (apply-rules rules) node))

(define (fixed-point next arg)
(let loop ((prev arg)
                (current (next arg)))
        (if (node=? prev current)
        current
        (loop current (next current)))))

(simplify node)
)))


;;;-----------------------------------------------------------------------
;;; General rule manipulation

(define (apply-rules rules)
(lambda (node)
(rules node)))

(define (compose-rules . procs)
(rec-reduce o identity procs))
```

```
(define (identity x) x)

(define (o f g)
(lambda (x)
(f (g x))))

(define (rec-reduce op id lst)
(let recur ((lst lst))
(if (null? lst)
id
(op (car lst) (recur (cdr lst))))))

;;;-------------------------------------------------------------------------
;;; Sample program

(define sample-program
'((swap exec swap exec) (1 sub) swap (2 mul) swap 3 swap exec))


;;;-------------------------------------------------------------------------
;;; PostFix Syntactic 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
( (int->sexp n)      ($int n)                      )
( (list->sexp lst)   ($seq (pf-sequence lst))        )
( 'pop              ($pop)                      )
( 'swap             ($swap)                      )
( 'exec             ($exec)                      )
( 'sel              ($sel)                      )
( 'dup              ($dup)                      )
;; Below, arithop and relop operations are functions, not symbols!
( 'add              ($arithop +)                  )
( 'sub              ($arithop -)                  )
( 'mul              ($arithop *)                  )
( 'div              ($arithop quotient)            ) ; integer division
( 'lt               ($relop   <)                  )
( 'eq               ($relop   =)                  )
( 'gt               ($relop   >)                  )
( _                 (error "Unrecognized command"
                     sexp)                  )
))

;;;------------------------------------------------------------------------
;;; Unparsing

(define (pf-unprogram pgm)
(match pgm
(($prog cmds) (pf-uncommands cmds))))

(define (pf-uncommands cmds)
(map pf-uncommand cmds))

(define (pf-uncommand cmd)
(match cmd
(($int i) i)
(($seq cmds) (pf-uncommands cmds))
(($pop) 'pop)
(($swap) 'swap)
(($dup) 'dup)
(($sel) 'sel)
(($exec) 'exec)
(($arithop op)
(cond ((eq? op +) 'add)
((eq? op -) 'sub)
```

```
      b.   ((eq? op *) 'mul)
      c.   ((eq? op quotient) 'div)))
102.       (($relop op)
103.       (cond ((eq? op <) 'lt)
      a.   ((eq? op =) 'eq)
      b.   ((eq? op >) 'gtl)))
104.       ))
```