```
; file: bingo.scm
; purpose: determine winning bingo cards given a list of numbers called
;       and a list of numbers on each card.
; chad c d clark < frink _at_ superfrink _dot_ net >
; date: 08 May 2003
; changes:
;  - 12 may 2003 : Made more efficient by replacing list? calls with pair? .
;       This changed some functions from O(n^2) to O(n).


; main() reads the input file from stdin and calls evaluate-cards() on it.
(define
  main
  (lambda ()
    (evaluate-cards (read))
) )


; evaluate-cards() deals with the details of preparing the input, etc.
(define
  evaluate-cards
  (lambda (ls)
    ;(display "evaluate-cards called\n")
    (if (good-input? ls)
      (begin
        (analyze-cards
          (cdr (car ls))        ; numbers
          (cdr ls)              ; card lines
        )
        #t
      )
      (begin
        (display "The data file is not correctly formated.\n")
        (display "No analysis could be done.\n")
        #f
      )
) ) )


; good-input?() returns #f if the input is not in the expected format,
; #t otherwise.
(define
  good-input?
  (lambda (ls)
    ;(display "good-input called\n")
    (cond
      ((not (number-line? (car ls)))
        (display "Invalid \"Number\" list.\n")
        #f )
      ((not (card-list? (cdr ls)))
        (display "Invalid \"Card\" list.\n")
        #f )
      (else #t)
) ) )


; number-line?() returns #t if the list 'ls' is of the form:
;   "Numbers" . <a list of numbers>
; otherwise #f is returned.
(define
  number-line?
  (lambda (ls)
    ;(display "number-line? called\n")
    (cond
      ((null? ls) #f)
```

```
      ((not (pair? ls)) #f)
      ((not (string? (car ls))) #f)
      ((not (string=? "Numbers" (car ls))) #f)
      ((not (list-of-numbers? (cdr ls))) #f)
      (else #t)
) ) )


; card-list?() returns #t if each item in the list 'ls' is a valid card list,
; #f is returned otherwise.
(define
  card-list?
  (lambda (ls)
    ;(display "card-list? called\n")
    (cond
      ((null? ls) #t)
      ((not (pair? ls)) #f)
      ((not (card-line? (car ls))) #f)
      (else (card-list? (cdr ls)))
) ) )


; card-line?() returns #t if the list 'ls' is of the form:
;   <a string> . <a list of numbers>
; otherwise #f is returned.
(define
  card-line?
  (lambda (ls)
    ;(display "card-line? called\n")
    (cond
      ((null? ls) #f)
      ((not (pair? ls)) #f)
      ((not (string? (car ls))) #f)
      ((not (list-of-numbers? (cdr ls))) #f)
      (else #t)
) ) )


; list-of-numbers?() returns #t if the list 'ls' is a proper list consisting
; entirely of numbers.
(define
  list-of-numbers?
  (lambda (ls)
    (cond
      ((null? ls) #t)
      ((not (pair? ls)) #f)
      ((not (number? (car ls))) #f)
      (else (list-of-numbers? (cdr ls)))
) ) )


; the core of the program, analyze-cards() calls for a report of each cards
; remaining numbers to be displayed.
(define
  analyze-cards
  (lambda (nums cards)
    (cond
      ((null? cards) ())        ; all cards done  :)
      (else
        (card-report
          (car (car cards))                    ; card name
          (set-less-set (cdr (car cards)) nums) ; numbers on card that have
                                                ; not been called yet
        )
        (analyze-cards nums (cdr cards))        ; do next card
```

```
) ) ) )


; set-less-elem() returns a list of all elements in the list 'set' except
; for the element 'elem'.
(define
  set-less-elem
  (lambda (set elem)
    (cond
      ((null? set) ())
      ((not (pair? set)) #f)
      ((= (car set) elem) (set-less-elem (cdr set) elem))
      (else (cons (car set) (set-less-elem (cdr set) elem)))
) ) )


; set-less-set() returns a list of all elements in the list 's1' except
; for the elements in the list 's2'.
(define
  set-less-set
  (lambda (s1 s2)
    (cond
      ((null? s2) s1)
      ((null? s1) ())
      (else
        (set-less-set
          (set-less-elem s1 (car s2))
          (cdr s2)
) ) ) ) )


; card-report() prints a summary of a card and it's remaining numbers.
(define
  card-report
  (lambda (card ls)
    (display card)
    (display " has ")
    (display (length ls))
    (display " numbers left: ")
    (write ls)
    (display ".\n")
) )


; i) call main() to start things going:
;    (display "Start of Analysis:\n")
;    (main)
;    (display "End of Analysis:\n")
;
; or ii) call with the input from a certain file:
;        (with-input-from-file "in-file" main)
```