Data Definitions

;; An entry is a structure
;; (make-entry Na Nu)
;; where Na is a symbol and Nu is a number
(define-struct entry (name number))

;; address-book : list of entry
;; keep track of the current address book entries
(define address-book empty)

Contracts, Purposes, Headers

;; lookup-number : symbol address-book ➝ (number or false)
;; Purpose: returns the phone number associated with the symbol,
;; or false if the symbol is not found
(define (lookup-number name) ...)

;; add-to-address-book : symbol number ➝ true
;; Purpose: adds the given name & number to the address book
(define (add-to-address-book name phone) ...)
;; lookup-number : symbol → (number or false)
;; Purpose: returns the phone number associated with the symbol,
;; or false if the symbol is not found
(define (lookup-number name)
  (local [(define matches
    (filter (lambda (an-entry)
      (symbol=? name (entry-name an-entry)))
    address-book))]
    (cond
      [(empty? matches) false]
      [else (entry-number (first matches))])))

;; add-to-address-book : symbol number → true
;; Purpose: adds the given name & number to the address book
(define (add-to-address-book name num)
  (begin
    (set! address-book
      (cons (make-entry name num) address-book))
    true))
;;; update-address: symbol number ⇒ void
;;; Purpose: given a name and number, updates the phone number
;;; for that name
(define (update-address name num)
  (local [(define updated-book
            (map (lambda (entry)
                  (cond
                    [(symbol=? (entry-name entry) name)
                      (make-entry name num)]
                    [else entry]))
            address-book))]
    (set! address-book updated-book))
))
;; update-address-book! : symbol number  →  void
;; Purpose: given a name and number, updates the phone number
;;          for that name
;; Effect: changes the phone number stored with the given name
;;         in address book
(define (update-address-book! name new-num)
  (local [(define (helper! a-book)
                (cond [(empty? helper) void]
                      [(else
                          (cond [(symbol=? name
                                   (entry-name (first a-book)))
                                (set-entry-phone! (first a-book) new-num)]
                              [else (helper! (rest a-book))]))]))
           (helper! address-book))))