```lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; address-book.lisp
;
; Created on February 14, 2013 by Matthew A. Crist.
;
; Purpose:
; This file contains the functions that are essential to acquiring address
; book information and storing entries into the address book.
;
; CHANGE LOG:
; ----------------------------------------------------------------------
; 2013-04-07    - Adjusted isInAddressBook to only check domain and name.
; 2013-03-17    - Added password field to address book for verification.
; 2013-02-19    - Predicate isInAdressBook was making incorrect reference
;                 to variable address when checking endp for recursion.
;                 this caused stack overflow.  Corrected to endp
;                 addressBook.
;
; 2013-02-19    - Added predicate test to addAddress function that would
;                 determine if the address was already in the address
;                 book.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(in-package "ACL2")

; (getAddress tokens)
; Acquires the address structure for an address entry that exists in the
; address book. (domain name) form.
; tokens - the tokenized XML string that will be used to extract contact
;          information.
(defun getAddress (tokens)
  (if (endp tokens)
      nil
      (if (equal "</address>" (caar tokens))
          nil
          (if (equal "<domain>" (caar tokens))
              (cons (caadr tokens) (getAddress (cdddr tokens)))
              (if (equal "<name>" (caar tokens))
                  (cons (caadr tokens) (getAddress (cdddr tokens)))
                  (if (equal "<password>" (caar tokens))
                      (cons (caadr tokens) (getAddress (cdddr tokens)))
                      (getAddress (cdr tokens))))))))

; (parseAddresses tokens)
; Acquires all the addresses that are present in the tokenized XML string
; and returns the list of the addresses to the requestor.
; tokens - the tokenized XML string that will be used to extract address
;          information.
(defun parseAddresses (tokens)
  (if (endp tokens)
      nil
      (if (equal "<address>" (caar tokens))
          (cons (getAddress (cdr tokens)) (parseAddresses (cdr tokens)))
          (parseAddresses (cdr tokens)))))

; (getAddressBook tokens)
; Acquires the address book - point on entry for address book acquisition.
; tokens - the tokenized XML string that will be used to extract the
;          address book.
(defun getAddressBook (tokens)
  (if (endp tokens)
      nil
      (let* ((addresses (parseAddresses tokens)))
        addresses)))

; (addressXML address)
; Generates the XML form for an address from the address structure.
; address - the address structure in the form of (domain name).
(defun addressXML (address)
  (if (endp address)
      nil
      ; Domain tag <domain>#PCDATA</domain>
      (let* ((domain (concatenate 'string
                     "<domain>" (car address) "</domain>"))
             ; Name tag <name>#PCDATA</name>
             (name   (concatenate 'string
             "<name>" (cadr address) "</name>"))
             ; Address tag <address>[domain][name]</address>
             (password (concatenate 'string
                     "<password>" (caddr address) "</password>"))
             (address (concatenate 'string
                     "<address>" domain name password "</address>")))
        address)))

; (addressBookXML addressBook)
; Generates the non header portion of the XML address book output.
; addressBook - the address book structure to be converted to XML.
(defun addressBookXML (addressBook)
  (if (endp addressBook)
      nil
      (cons (addressXML (car addressBook))
            (addressBookXML (cdr addressBook)))))

; (getAddressBookXML addressbook)
; Converts the address book data structure into XML that can be stored to
; a document.
; addressBook - the address book stuctures that is to be converted to XML.
(defun getAddressBookXML (addressBook)
    (if (endp addressBook)
        nil
        (let* ((xml (append (append (list
    "<?xml version='1.0'?>"
    "<!DOCTYPE addresses SYSTEM '../../dtd/address-book.dtd'>"
    "<addresses>")
        (addressBookXML addressBook))
    '("</addresses>"))))
            xml)))

; (isInAddressBook addressBook address)
; Predicate to determine if an address is in the address book.
; addressBook - the address book to use to determine if an address is
;               contained in this address book.
; address     - the address in which we are to be locating.
(defun isInAddressBook (addressBook address)
  (if (endp addressBook)
      nil
      (if (and (equal (caar addressBook) (car address))
               (equal (cadar addressBook) (cadr address)))
          t
          (isInAddressBook (cdr addressBook) address))))

; (addAddress addressBook address)
; Appends an address onto the end of the address book structure.  If the
; user is already in the address book, then the return is the original
; address book.
; addressBook - the address book in which to add the address.
; address     - the address to add to the address book.
(defun addAddress (addressBook address)
  (if (equal (isInAddressBook addressBook address) nil)
      (append addressBook (list address))
      addressBook))

; (removeAddress addressBook address)
; Removes an address from the address book.
; addressBook - the address book in which the address supposedly exists.
; address     - the address to remove from the address book.
(defun removeAddress (addressBook address)
  (if (endp addressBook)
      nil
      (if (equal (car addressBook) address)
          (cdr addressBook)
          (cons (car addressBook)
                (removeAddress (cdr addressBook) address)))))
```