```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; address-book_tests.lisp                                                    ;
; Created by Matthew A. Crist on April 7, 2013 (augmented)                   ;
; This file holds the tests for the logic on the address-book data           ;
; structure.  This file should be tested using the Racket IDE with           ;
; dracula.                                                                   ;
;                                                                            ;
; CHANGE LOG:                                                                ;
; -------------------------------------------------------------------------- ;
; 2013-04-07 - Added heading section for this file.  File was created        ;
;              previously, but was never formalized in documentation.        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(include-book "address-book")
(include-book "doublecheck" :dir :teachpacks)
(include-book "testing" :dir :teachpacks)

; Theorem:
; If the address is in the address book, then adding the address to
; the address book results in the same address book structure.
;(defthm address-is-in-book-adding-returns-original-book-thm
;  (implies (and (listp aB)
;                (isInAddressBook aB a))
;           (equal aB (addAddress aB a)))
;  :rule-classes (:rewrite :forward-chaining))

; Test on above theorem
(defproperty address-is-in-book-adding-returns-original-book-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (isInAddressBook addressBook address)
           (equal addressBook (addAddress addressBook address)))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check Expects on above theorem
(check-expect (addAddress '((1 2 3)) '(1 2 3)) '((1 2 3)))

; Theorem:
; If the address is not in the address book, then adding the address
; to the address book will return an address book with the length of
; the original address book + 1. (admits)
;(defthm address-is-not-in-book-add-length-thm
;  (implies (and (listp addressBook)
;                (not (isInAddressBook addressBook address)))
;           (equal (+ (length addressBook) 1)
;                  (length (addAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))

; Test on the above theorem
(defproperty address-is-not-in-book-add-length-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   domain-not  :where (stringp domain-not)
               :value (random-string)
```

```
   name-not    :where (stringp name-not)
               :value (random-string)
   pass-not    :where (stringp pass-not)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   address-not :where (listp address-not)
               :value (list domain-not name-not pass-not)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (not (isInAddressBook addressBook address-not))
           (equal (+ (length addressBook) 1)
                  (length (addAddress addressBook address-not))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check-expects on the above theorem
(check-expect (length (addAddress '((1 2 3)) '(4 5 6))) 2)


; Theorem:
; If the address is in the address book, then removing the address from
; the address book will return an address book with the length of the
; original address book - 1.
;(defthm address-is-in-book-remove-length-thm
;  (implies (and (listp addressBook)
;                (not (equal address nil))
;                (not (equal (length addressBook) 0))
;                (isInAddressBook addressBook address))
;           (equal (- (length addressBook) 1)
;                  (length (removeAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))


; Tests on the above theorem
(defproperty address-is-in-book-remove-length-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (isInAddressBook addressBook address)
           (equal (- (length addressBook) 1)
                  (length (removeAddress addressBook address))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check-Expects on above example
(check-expect (length (removeAddress '((1 2 3) (4 5 6)) '(1 2 3))) 1)

; Theorem:
; If the address is not in the address book, then remove the address from
; the address book results in the original address book (lengths are
; equivalent).
;(defthm address-is-not-in-address-book-remove-returns-original-book-thm
;  (implies (and (listp addressBook)
;                (not (isInAddressbook addressBook address)))
;           (equal (length addressBook)
;                          (length (removeAddress addressBook address))))
;  :rule-classes (:rewrite :forward-chaining))
```

```
; Tests on the above theorem
(defproperty address-is-not-in-address-book-remove-returns-original-book-tst
  (domain      :where (stringp domain)
               :value (random-string)
   name        :where (stringp name)
               :value (random-string)
   pass        :where (stringp pass)
               :value (random-string)
   domain-not  :where (stringp domain-not)
               :value (random-string)
   name-not    :where (stringp name-not)
               :value (random-string)
   pass-not    :where (stringp pass-not)
               :value (random-string)
   address     :where (listp address)
               :value (list domain name pass)
   address-not :where (listp address-not)
               :value (list domain-not name-not pass-not)
   addressBook :where (listp addressBook)
               :value (list address))
  (implies (not (isInAddressbook addressBook address-not))
           (equal (length addressBook)
                  (length (removeAddress addressBook address-not))))
  :rule-classes (:rewrite :forward-chaining))

(check-properties)

; Check expect on the above test
(check-expect (length (removeAddress '((1 2 3)) '(4 5 6))) 1)
```