



Ejercicios para resolver en papel

5.1 Estructuras y Listas

Recursión Simple

Sistema de contraseñas

En un sistema de contraseñas de acceso a un sistema operativo, cada persona cuenta con un nombre identificador único, una contraseña de acceso y un nivel de permisos en el sistema. Una persona puede tener permisos de administración (especiales) del sistema; o bien, tener permisos de uso básico del sistema. El programa que maneja las personas usuarias del sistema, sus contraseñas y sus permisos representa el registro de cada persona mediante la siguiente estructura.

```
(define-struct Usr [login pass permiso])

;Usr es (String, String, Number)
;Un elemento Usr representa el registro de una persona con acceso al sistema
; operativo donde
;login: es el nombre identificador de la persona,
;pass: es la contraseña de acceso,
;permiso: es el identificador de los permisos en el sistema que tiene la persona
; Si tiene permiso de administración, el valor es 0; en otro caso es 1.

; 0 es el identificador de permisos de administración del sistema
(define ADMIN 0)

;1 es el identificador de permisos de uso básico del sistema
(define USRPERMISO 1)


;Constantes que pueden ser usadas para casos de prueba.
(define ANA (make-Usr "ana" "12345678" ADMIN))
(define LUIS (make-Usr "luis" "12345678" USRPERMISO))
(define MARTA (make-Usr "marta", "R34dlsoA" ADMIN))
(define L1 (list ANA LUIS))
(define L2 (list ANA LUIS MARTA))
```

 **Ejercicio 1.** Complete el siguiente diseño dando la definición de la función `cambioClave`.

```
;cambioClave: Usr String ->Usr

;Esta función recibe el registro de una persona usuaria del sistema y una contraseña.
;Si la contraseña recibida es diferente a la que tiene el registro y tiene 8
; o más caracteres, la función cambia la contraseña en el registro y lo devuelve;
; en caso contrario, devuelve el registro sin modificaciones.

(check-expect (cambioClave ANA "aaa") ANA)
(check-expect (cambioClave ANA "12345678") ANA)
(check-expect (cambioClave ANA "abcdefghi") (make-Usr "ana" "abcdefghi" ADMIN))
```

 **Ejercicio 2.** Complete el siguiente diseño dando la definición de la función `agregaUsr`.
No defina funciones auxiliares ni use funciones del lenguaje como `member`?

```
;agregaUsr: String String Number List(Usr) ->List(Usr)

;Esta función agrega un nuevo registro de una persona con acceso al sistema.
;Para esto recibe un nombre identificador de la persona, una contraseña de acceso,
; un identificador de permisos y la lista de registros de personas con acceso
; al sistema. Si no existe un registro con el identificador recibido, agrega uno
; nuevo a la lista, con la correspondiente información, y la devuelve;
; en caso contrario, no lo agrega y devuelve la lista original.



(check-expect (agregarUsr "marta" "R34dlsoA" ADMIN L1) L2)
(check-expect (agregarUsr "luis" "R34dlsoA" ADMIN L1) L1)
(check-expect (agregarUsr "marta" "R34dlsoA" ADMIN empty) (list MARTA))
```

 **Ejercicio 3.** Complete el siguiente diseño dando la definición de `esAdmin?`.

```
;esAdmin?: Usr ->Boolean

;Esta función recibe el registro de una persona usuaria del sistema y determina
; si tiene permisos de administración.


(check-expect (esAdmin? ANA) #t)
(check-expect (esAdmin? LUIS) #f)
```

 **Ejercicio 4.** Complete el siguiente diseño dando la definición de la función `cantAdmin`.
Sugerencia: use `esAdmin?` del  Ejercicio. 3.

```
;cantAdmin: List(Usr) ->Number

;Esta función recibe una lista de registro de personas usuarias del sistema y
; devuelve la cantidad de personas que tienen permisos de administración.

(check-expect (cantAdmin L2) 2)
(check-expect (cantAdmin (list LUIS)) 0)
(check-expect (cantAdmin empty) 0)
```

 **Ejercicio 5.** Complete el siguiente diseño dando la definición de la función `eliminaUsr`.

```
;eliminaUsr: String List(Usr) ->List(Usr)

;Esta función recibe un identificador de persona en el sistema y la lista de
; registros de personas con acceso al sistema. Si existe un registro con tal
; identificador lo elimina de la lista y devuelve la lista resultante; en caso
; contrario devuelve la lista sin modificaciones.

(check-expect (eliminaUsr "marta" L2) L1)
(check-expect (eliminaUsr "daniel" L2) L2)
(check-expect (eliminaUsr "marta" empty) empty)
```

 **Ejercicio 6.** Complete el siguiente diseño dando la definición de la función `bloquearClaves`.

```
;bloquearClaves: List(Usr) ->List(Usr)
;Esta función recibe una la lista de registros de personas con acceso al sistema,
; cambia la clave de cada registro a "nula" y devuelve la lista modificada.
(check-expect (bloquearClaves L1) (list (make-Usr "ana" "nula" ADMIN)
                                         (make-Usr "luis" "nula" USRPERMISO)))
(check-expect (bloquearClaves empty) empty)
```

Agregando un poco de dificultad a la recursión simple

 **Ejercicio 7*.** De la definición de la siguiente función.

```
;intercala: Lista(Any) Lista(Any) ->Lista(Any)
;Esta función intercala los elementos de dos listas dadas.
(check-expect (intercala (list 1 2 3) (list "A" "B")) (list 1 "A" 2 "B" 3))
(check-expect (intercala (list "A" "B") (list 1 2 3) ) (list "A" 1 "B" 2 3))
```

 **Ejercicio 8*.** De la definición de la siguiente función.

```
;ultimo : List (Any) ->Any
;Devuelve el último elemento de una lista NO vacía.
(check-expect (ultimo (list 1 2 3)) 3)
(check-expect (ultimo (list 1 2)) 2)
(check-expect (ultimo (list 1)) 1)
```

 **Ejercicio 9*.** De la definición de la siguiente función.

```
;listaCapicua:List(Number) ->Boolean
;Determina si la secuencia de una lista de números de un dígito forman un número
capicúa.
(check-expect (listaCapicua (list 1)) #t)
(check-expect (listaCapicua (list 1 1)) #t)
(check-expect (listaCapicua (list 1 2 3 2 1)) #t)
(check-expect (listaCapicua (list 1 2)) #f)
(check-expect (listaCapicua (list 1 2 3)) #f)
(check-expect (listaCapicua (list 1 2 3 4 1)) #f)
(check-expect (listaCapicua (list 1 2 3 4 5 6 7 8)) #f)
```