Nombre:_____

Lenguajes y Paradigmas de Programación
Examen recuperación parcial 2 convocatoria C4 2014-15

Ejercicio 1 (2,5 puntos)

a) (0,5 puntos) Supongamos la siguiente función recursiva:

```
(define (suma lista)
   (if (null? (cdr lista))
       (car lista)
       (+ (car lista) (suma (cdr lista)))))
```

Escribe la traza de las llamadas recursivas que se generan cuando se evalúa la expresión:

```
(suma '(1 2 4 8))
```

¿Cuántas llamadas se quedan en espera? ¿Cuál es el coste temporal y espacial de la ejecución?

b) (0,5 puntos) Escribe una versión iterativa de la función anterior que utilice recursión por la cola.

c) (1,5 puntos) Escribe **utilizando recursión por la cola** la función `(elimina-elem elem lista)` que recibe como argumento un elemento y una lista y devuelve otra lista en la que se han eliminado las ocurrencias de `elem`

Ejemplo:

```
(elimina-elem 5 '(8 3 5 2 5 1)) ⇒ (8 3 2 1)
```

Ejercicio 2 (2,5 puntos)

a) (0,5 puntos) Define la función `(primero lista)` que devuelve la primera hoja de una lista estructurada. Suponemos que la lista no es vacía.

Ejemplo:

```
(primero '(((1) 7) 2 (3 4) ((10) 6))) ⇒ 1
```

b) (0,75 puntos) Define la función `(ultimo lista)` que devuelve la última hoja de una lista estructurada. Suponemos que la lista no es vacía.

Ejemplo:

```
(ultimo '((1 7) 2 (3 4) ((10) 6))) ⇒ 6
```

c) (1,25 puntos) Define la función recursiva `(ordenada? lista)` que comprueba si una lista estructurada de números tiene sus hojas ordenadas. Puedes utilizar las funciones definidas en los apartados anteriores.

Ejemplos:

```
(ordenada? '(((1 2 (3)) 4 (5 (6)))))
⇒ #t
(ordenada? '(((1 2 (7)) 4 (5 (6)))))
⇒ #f
```

Ejercicio 3 (2,5 puntos)

a) (1,25 puntos) Escribe la función `(sustituye-elem lista elem-old elem-new)` que recibe como argumentos una lista estructurada y dos elementos, y devuelve otra lista con la misma estructura, pero en la que se ha sustituido las ocurrencias de `elem-old` por `elem-new`

Ejemplo:

```
(define lista  '(1 2 (3 4 (5 3)) 3 (8 (3) 10)))
(sustituye-elem  lista  3  0)
⇒ (1 2 (0 4 (5 0)) 0 (8 (0) 10))
```

b) (1,25 puntos) Define la función `(mayor-dato-tree tree)` que devuelva el mayor número de un árbol. Puedes hacerlo con funciones de orden superior o con recursión.

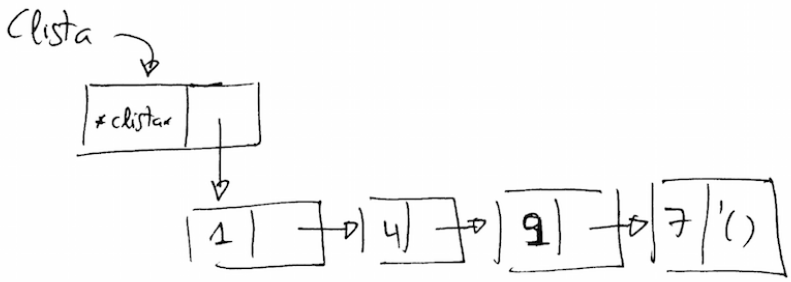
```
(define arbol
    '(4 (5 (2) (3)) (10) (20 (40) (15 (13) (14)) (17) (19 (18)))))
(mayor-dato-tree arbol) ⇒ 40
```

Ejercicio 4 (2,5 puntos)

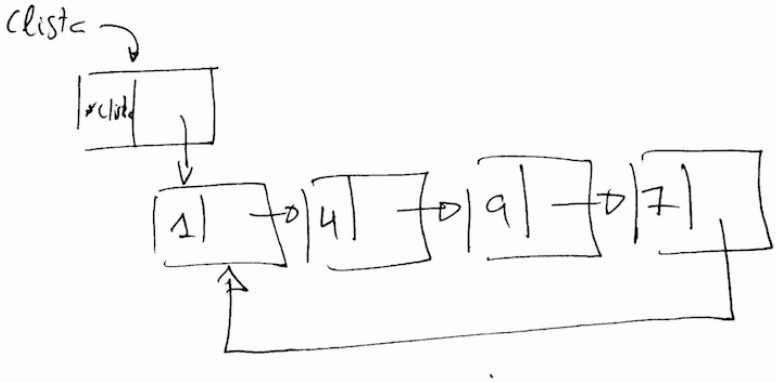
a) (1,25 puntos) Implementa el procedimiento mutador `(crear-lista-circular! clista)` que reciba una lista con cabecera y devuelva la correspondiente lista circular: el siguiente al último elemento será el primer elemento

Ejemplo:

```
(define clista '(*clista* 1 4 9 7))
```




```
(crear-lista-circular! clista)
```



b) (1,25 puntos) Define `(existe-elemento? elem clista)` que recibe un elemento y una lista circular y devuelve `#t` o `#f` si se encuentra o no el elemento en la lista.

Ejemplo:

```
(existe-elemento? 5
   (crear-lista-circular! '(*clist* 1 4 9 7)) ⇒ #f
```