Práctica 3: Parejas, datos compuestos y listas

Para entregar la práctica debes subir a Moodle un único fichero `practica03.scm` con los ejercicios resueltos y las pruebas que comprueban su correcto funcionamiento. En el fichero debes incluir la imagen que pedimos en el ejercicio 2 (copiando y pegando del portapapeles), **no debes subir un fichero comprimido**.

Ejercicio 1

a) Empezamos definiendo unas funciones auxiliares que vamos a necesitar. Construye las funciones `(inc-izquierda pareja)` y `(inc-derecha pareja)` definidas de la siguiente forma:

- `(inc-izquierda pareja)` : recibe una pareja de números y devuelve una nueva nueva pareja en la que se ha sumado 1 a la parte izquierda de `pareja`
- `(inc-derecha pareja)` : recibe una pareja de números y devuelve una nueva pareja en la que se ha sumado 1 a la parte derecha de `pareja`

Por ejemplo:

```
(inc-izquierda '(12 . 20)) ⇒ (13 . 20)
(inc-derecha '(12 . 20)) ⇒ (12 . 21)
```

b) Define una función recursiva `(cuenta-pares-impares lista-num)` que devuelva una pareja cuya parte izquierda sea el número de números pares de `lista-num` y la parte derecha el número de sus números impares. En el caso en que no existan números pares o impares en `lista-num` se devolverá 0 en la parte correspondiente de la pareja. La función recursiva debe utilizar las funciones `inc-izquierda` e `inc-derecha` definidas en el apartado anterior.

```
(cuenta-pares-impares '(1 2 3 4 5 6)) ⇒ (3 . 3)
(cuenta-pares-impares '(1 1 1 1 1 1)) ⇒ (0 . 6)
```

c) Define una nueva función recursiva `(suma-pares-impares lista-num)` que devuelva una pareja cuya parte izquierda sea la suma de todos los números pares de la lista y la parte derecha la suma de todos sus números impares. En el caso en que ningún número par o impar deberá devolver 0. Puedes definir las funciones auxiliares que necesites.

Ejemplos:

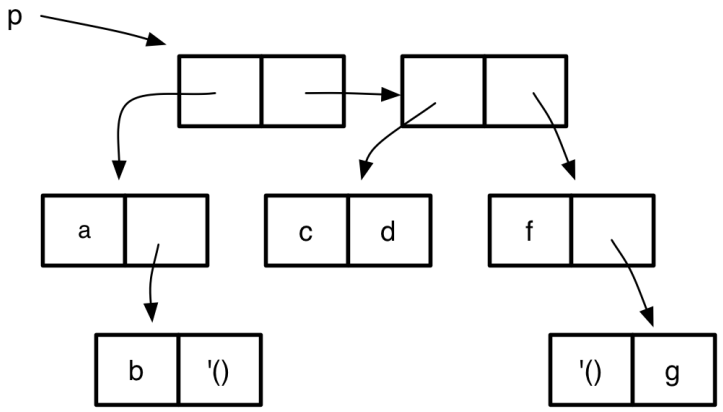
```
(suma-pares-impares '(1 2 3 4 5 6)) ⇒ (12 . 9)
(suma-pares-impares '(1 3 5 7)) ⇒ (0 . 16)
```

Ejercicio 2

a) Dibuja el diagrama *box & pointer* de las siguientes expresiones, explica si `p1`, `p2` y `p3` son listas y cuántos elementos tienen (en el caso en que lo sean). Incluye el diagrama en el mismo fichero `practica03.scm`, copiando y pegando del portapapeles.

```
(define p1 (cons 'b
                 (cons 'c '())))
(define p2 (cons 'd
                 (cons 'e 'f)))
(define p3 (cons 'a
                 (cons p1
                       (list p2 (cons 'g '())))))
```

b) Dado el siguiente *box & pointer*, escribe una expresión en Scheme que lo genere usando sólo llamadas a `cons` y explica si la expresión resultante es una lista y, en el caso en que la sea, cuántos elementos tiene:



Ejercicio 3

Escribe la función recursiva `(mayor-que n lista-nums)` que recibe un número `n` y un lista de números y devuelve una lista con los booleanos resultantes de comparar cada número de la lista con `n`.

Ejemplo:

```
(mayor-que 10 '(1 23 4 -1 12 11 10)) ⇒ (#f #t #f #f #t #t #f)
```

Ejercicio 4

Define la función recursiva `(puntos3d-a-2d lista)` que reciba una lista cuyos elementos representan puntos 3d con coordenadas (x y z) y devuelva una lista de puntos 2d en la que se ha eliminado la componente z de cada punto 3d.

Los puntos 3d vienen representados por listas de 3 elementos, mientras que los puntos 2d

deben ser representados por parejas.

Ejemplo:

```
(puntos3d-a-2d '((1 2 3) (4 5 6) (7 8 9) (10 11 12)))
⇒ ((1 . 2) (4 . 5) (7 . 8) (10 . 11))
```

Ejercicio 5

Define la función recursiva `(anyade-si-suma>n lista1 lista2 n)` que reciba dos listas y un número `n`. Deberá devolver una nueva lista donde se hayan añadido los elementos de ambas sólo si la suma de los elementos en la misma posición es mayor o igual que `n`.

```
(define l1 '(1 2 3 4 3 4 5 6 7))
(define l2 '(5 1 2 1 2 1 0 2 0))
(anyade-si-suma>n l1 l2 6)
⇒ (1 5 6 2 7 0)
```

Ejercicio 6

Define la función recursiva `(posiciones-pares lista)` que devuelva una lista con los elementos situados en las posiciones 0, 2, 4, … de la lista que se pasa como argumento:

```
(posiciones-pares '(a b c d e f)) ⇒ (a c e)
```

Ejercicio 7

Escribe la función recursiva `(expande lista-parejas)` que reciba una lista de parejas que contienen un dato y un número y devuelva una lista donde se hayan “expandido” las parejas, añadiendo tantos elementos como el número que indique cada pareja.

Ejemplo:

```
(expande (list (cons 'a 4) (cons "hola" 2) (cons #f 3)))
⇒ (a a a a "hola" "hola" #f #f #f)
```
