Práctica 6: Listas estructuradas y árboles

En esta práctica vamos a trabajar con listas estructuradas y árboles. Debes comenzar incluyendo en el fichero las siguientes funciones vistas en teoría que definen su barrera de abstracción:

Funciones sobre listas

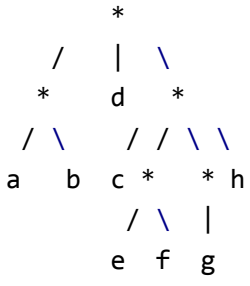
- `(hoja? lista)`

Funciones sobre árboles

- `(make-tree dato hijos)`
- `(make-hoja-tree dato)`
- `(dato-tree tree)`
- `(hijos-tree tree)`
- `(hoja-tree tree)`

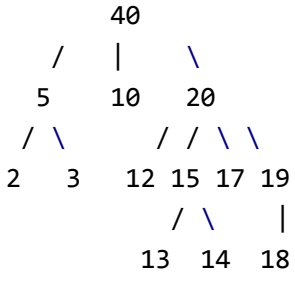
Ejercicio 1

a) Escribe la **lista estructurada** correspondiente al siguiente *pseudo-árbol*:




```
(define lista '(________))
```

b) Escribe la **lista estructurada** correspondiente al siguiente árbol:



```
(define arbol '(________))
```

Ejercicio 2

a) Define los procedimientos `(suma-lista lista)` y `(suma-tree tree)` que calculan la suma de todos los elementos de la lista estructurada o el árbol que le pasamos como parámetro. Suponemos que son listas o árboles formados por números. Recuerda que un árbol es una lista estructurada, pero que no siempre las listas estructuradas son árboles.

```
(define lista '(1 (2 (3 ((((4)))) 5) 6)))
(define tree '(3 (5 (6) (7)) (4)))
(suma-lista lista) ⇒ 21
(suma-tree tree) ⇒ 25
(suma-lista tree) ⇒ 25
(suma-tree lista) ⇒ error
```

b) Define el procedimiento `(aplana lista)` que tome una lista estructurada como parámetro y devuelva una lista que contenga sus elementos.

Ejemplo:

```
(aplana '(1 (2 (3) (4 (5 (6) 7)))))
⇒ (1 2 3 4 5 6 7)
```

c) Escribe la función `(diff-listas l1 l2)` que tome como argumentos dos listas estructuradas con la misma estructura, pero con diferentes elementos, y devuelva una lista de parejas que contenga los elementos que son diferentes.

Ejemplos:

```
(dif-listas '(a (b ((c)) d e) f) '(1 (b ((2)) 3 4) f))
⇒ ((a . 1) (c . 2) (d . 3) (e . 4))
(diff-listas '() '())
⇒ ()
(diff-listas '((a b) c) '((a b) c))
⇒ ()
```

Ejercicio 3

a) Decimos que un árbol está ordenado cuando cumple las siguientes propiedades:

- El dato de la raíz es mayor que cualquiera de sus hijos
- Los datos de los las raíces de sus hijos están en orden creciente
- Todos los hijos son árboles que están ordenados

Escribe la función `(ordenado-tree? tree)` que compruebe si un árbol cumple las condiciones anteriores.

Ejemplos:

```
(ordenado-tree? '(10 (5) (7))) ⇒ #t
(ordenado-tree? '(50 (10 (4) (6) (8)) (25 (15)))) ⇒ #t
(ordenado-tree? '(10 (8) (7))) ⇒ #f
(ordenado-tree? '(6 (5) (7))) ⇒ #f
(ordenado-tree? '(50 (10 (4) (6) (11)) (25) (15))) ⇒ #f
```

b) Define la función recursiva `(calcula-tree tree)` que reciba como argumento un árbol que representa una expresión aritmética (con los símbolos `+`, `-`, `*` y `/` en los nodos y números en las hojas) y devuelva su resultado.

Ejemplo:

```
(calcula-tree '(+ (- (5) (2)) (3))) ⇒ 6
(calcula-tree '(* (- (2) (+ (3) (* (4) (- (6) (2)) (3)) (1))) (1))) ⇒ -50
```

Puedes ayudarte del siguiente procedimiento:

```
(define (operator op)
    (cond
        ((equal? op '+) +)
        ((equal? op '-) -)
        ((equal? op '*) *)
        ((equal? op '/) /)
        (else (error "Operador desconocido: " op))))
```

Ejercicio 4

a) Escribe la función `(nivel-hoja dato lista)` que recorra la lista estructurada buscando el dato y devuelva el nivel en que se encuentra. Suponemos que el dato está en la lista y no está repetido.

Ejemplos:

```
(nivel-hoja 'b '(a b (c))) ⇒ 1
(nivel-hoja 'b '(a (b) c)) ⇒ 2
(nivel-hoja 'b '(a c d ((b))) ⇒ 3
```

b) Escribe la función `(nivel-dato-tree dato tree)` que recorra el árbol y devuelve el nivel en que se encuentra el dato. Suponemos que el dato está en el árbol y no está repetido. Si el dato está en la raíz debe devolver 0.

Ejemplos:

```
(nivel-dato-tree 30 '(20 (18) (19 (30) (32)) (4))) ⇒ 2
(nivel-dato-tree 20 '(20 (18) (19 (30) (32)) (4))) ⇒ 0
```

Ejercicio 5

Define un procedimiento llamado `(transformar plantilla lista)` que reciba dos listas como argumento: la lista `plantilla` será una lista estructurada y estará compuesta por números enteros positivos con una estructura jerárquica, como (2 (3 1) 0 (4)). La segunda lista será una lista plana con tantos elementos como indica el mayor número de plantilla más uno (en el caso anterior serían 5 elementos). El procedimiento deberá devolver una lista estructurada donde los elementos de la segunda lista se sitúen (en situación y en estructura) según indique la plantilla.

Ejemplos:

```
(transformar '((0 1) 4 (2 3)) '(hola que tal estas hoy))
⇒ ((hola que) hoy (tal estas))
(transformar '(1 4 3 2 5 (0)) '(vamos todos a aprobar este examen))
⇒ (todos este aprobar a examen (vamos))
```
