Segundo parcial Lenguajes y Paradigmas de Programación
Curso 2017-18

Nombre:_____

Normas importantes

- **Los profesores no contestarán ninguna pregunta durante la realización del examen, exceptuando aquellas que estén relacionadas con algún posible error en el enunciado de alguna pregunta.**
- **Puedes definir y utilizar funciones auxiliares en aquellos ejercicios en los que sea posible.**
- **Debes utilizar las barreras de abstracción definidas para listas estructuradas, árboles y árboles binarios. No hace falta que definas su implementación.**
- **La duración del examen es de 2 horas.**

Ejercicio 1

a) (0,75 puntos) Dada la siguiente función, escribe su versión iterativa equivalente y escribe un `check-equal?` con un ejemplo que pruebe su funcionamiento en el que **debes utilizar una lista de 4 elementos**.

```
(define (bar lista)
   (if (null? lista)
      '()
      (append (bar (cdr lista))
              (list (car lista)))))

(check-equal? ____________________  ____________________ )
```

Iterativa equivalente:

b) (0,6 puntos) Supongamos que tenemos definida la función `(cuadrado w)` que dibuja un cuadrado de lado `w` que deja el cursor en el mismo punto y orientación que al comienzo. Enlaza cada código con el gráfico que genera la llamada `(f 10 10)`:

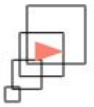
1.
```
(define (f lado incremento)
   (if (< lado 50)
      (begin
         (cuadrado lado)
         (turn 60)
         (move (* incremento -1))
         (turn -60)
         (f (+ lado incremento) incremento))))
```

2.
```
(define (f lado incremento)
   (if (< lado 50)
      (begin
         (cuadrado lado)
         (turn 180)
         (move incremento)
         (turn -180)
         (f (+ lado incremento) incremento))))
```

3.
```
(define (f lado incremento)
   (if (< lado 50)
      (begin
         (cuadrado lado)
         (turn 60)
         (move incremento)
         (turn -60)
         (f (+ lado incremento) incremento))))
```

4.
```
(define (f lado incremento)
   (if (< lado 50)
      (begin
         (cuadrado lado)
         (turn 180)
         (move (* incremento -1))
         (turn -180)
         (f (+ lado incremento) incremento))))
```

a.

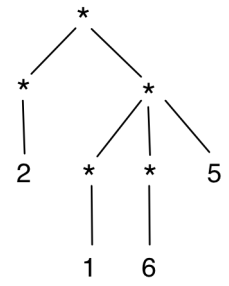


b.

c.

d.

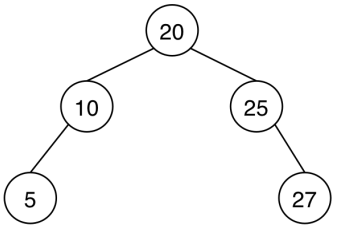
c) (0,9 puntos) Para cada una de las siguientes estructuras jerárquicas, escribe la sentencia que lo haya generado. Escribe también la instrucción para obtener el elemento que te piden utilizando la barrera de abstracción apropiada.



Lista estructurada (pseudoárbol):

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

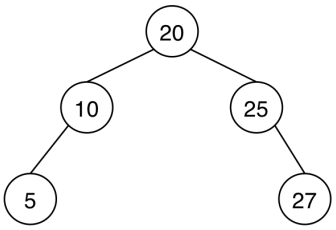
Obtén el elemento 6:



Árbol genérico:

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

Obtén el elemento 5:

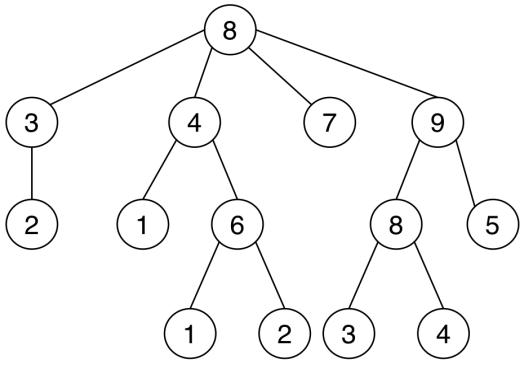


Árbol binario:

```
(define arbolb '(                                   ))
```

Obtén el elemento 5:

d) (0,75 puntos) Supongamos el siguiente árbol genérico que asignamos a la variable `arbol`



Y la siguiente función:

```
(define (g arbol)
    (if (hoja-arbol? arbol)
        (dato-arbol arbol)
        (+ (dato-arbol arbol)
           (g (cadr (hijos-arbol arbol))))))
```

Suponiendo la siguiente invocación a la función g:

```
(g (cadr (hijos-arbol arbol)))
```

d.1) ¿Qué árbol recibe como argumento **la primera llamada recursiva a g**? Dibújalo.

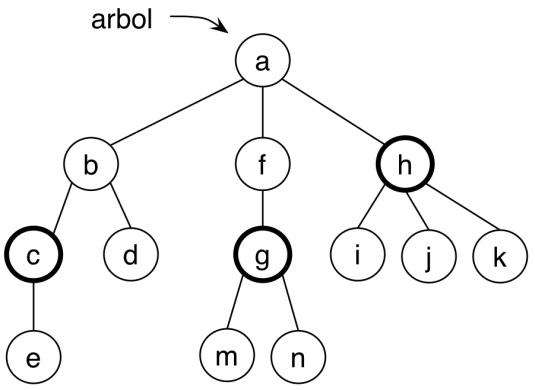
d.2) ¿Qué devuelve esta primera llamada recursiva?

Ejercicio 2

a) (1,25 puntos) Utilizando **funciones de orden superior**, escribe la función `(hijos-hoja? arbol)` que dado un árbol genérico, devuelve `#t` si todos los hijos del arbol son hojas y #f en caso contrario. **Si defines alguna otra función auxiliar, debes implementarla también con funciones de orden superior.**

Los nodos marcados en la figura son árboles cuyos hijos son hoja.

Ejemplo:




```
(check-false (hijos-hoja? arbol))
(check-true (hijos-hoja? (caddr (hijos-arbol arbol))))
```

b) (1,5 puntos) Utilizando la función del apartado anterior, escribe la función `(subhijos-hoja arbol)`, que dado un árbol genérico, devuelve una lista de parejas en donde cada pareja viene dada por el dato y el número de hijos de cada subárbol descendiente cuyos hijos sean hoja. Puedes usar recursión y/o funciones de orden superior. También puedes usar la función `length`.

Ejemplo:

```
(check-equal? (subhijos-hoja arbol) '((c . 1) (g . 2) (h . 3)))
```

Ejercicio 3 (1,25 puntos)

Define la función `(nivel-lista lista n)` que reciba una lista estructurada y un nivel, y devuelva una lista plana que contenga todos los elementos que se encuentran en ese nivel. Puedes usar recursión y/o funciones de orden superior.

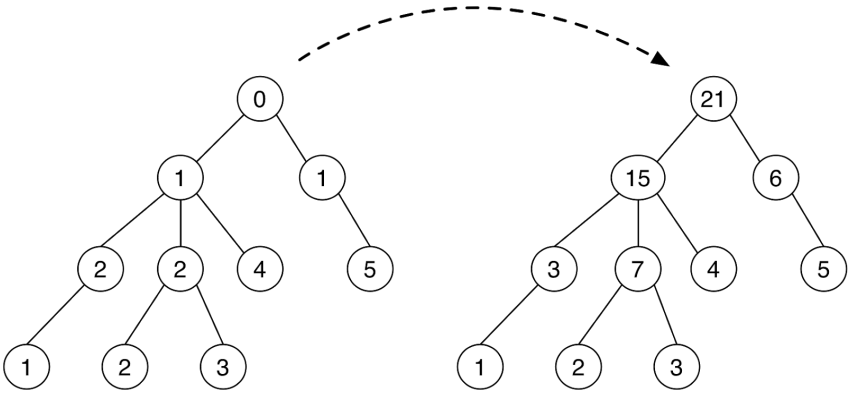
Ejemplo:

```
(define lista '(1 (2 (3)) (6) ((7 3)) (6 (5 5)(((4))))))
(nivel-lista lista 5) → '(4)
(nivel-lista lista 4) → '()
(nivel-lista lista 3) → '(3 7 3 5 5)
(nivel-lista lista 2) → '(2 6 6)
```

Ejercicio 4 (1,5 puntos)

Define una función `(suma-datos arbol)` que devuelva un árbol con la misma estructura que el que se le pase como argumento, pero en el que se suma al dato de cada nodo los datos de sus hijos tal y como se muestra en el ejemplo. La implementación debe ser lo más eficiente posible. Puedes usar recursión y/o funciones de orden superior.

Ejemplo:



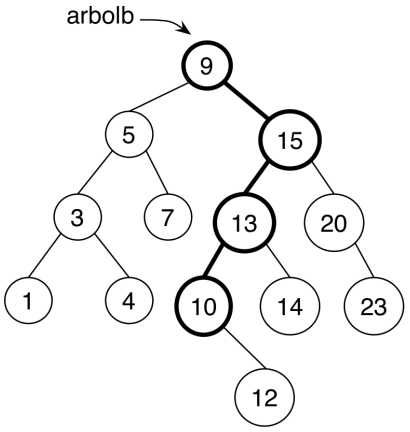
Ejercicio 5 (1,5 puntos)

Escribe la función `(camino-parcial arbolb nivel camino)` que tiene como parámetros un árbol binario, un nivel inicial y una lista de símbolos `'(i d)` que determina el camino a recorrer, en donde:

`'i` : indica que nos vamos por la izquierda
`'d` : indica que nos vamos por la derecha

La función debe devolver la lista con los datos de todos los nodos visitados indicados por el camino y a partir del nivel especificado. Suponemos que la lista define un camino correcto y que el nivel indicado también lo es (el camino pasa por ese nivel). El nivel 0 corresponde a la raíz.

Ejemplo:




```
(check-equal? (camino-parcial arbolb 2 '(d i i)) '(13 10))
(check-equal? (camino-parcial arbolb 1 '(d i i)) '(15 13 10))
(check-equal? (camino-parcial arbolb 0 '()) '(9))
```