Ejercicio 1.- Se dice que un árbol binario es "zurdo" en uno de estos tres casos:

- si es el árbol vacío; o
- si es una hoja; o
- si sus hijos izquierdo y derecho son los dos "zurdos" y el hijo izquierdo tiene más elementos que el hijo derecho.

Crear las operaciones necesarias para determinar si un árbol binario es "zurdo".

<u>**Solución:**</u>
Operaciones
num_elementos: a_bin→natural {Operación auxiliar}
func num_elementos (a:a_bin) **dev** n:natural
    si vacio?(a) **entonces** n←0
    sino n←1+num_elementos(izq(a))+num_elementos(der(a))
finfunc

zurdo: a_bin→bool
func zurdo (a:a_bin) **dev** b:bool
    si vacio?(a) V (vacio?(izq(a)) ∧ vacio?(der(a))) **entonces** b←T
        sino
        b←(num_elementos(der(a))< num_elementos(izq(a)))
        ∧ zurdo (izq(a)) ∧ zurdo(der(a))

Ejercicio 2.- Extender el TAD árboles binarios de naturales, añadiendo operaciones para:

a) Obtener la suma de todos los elementos que sean números pares del árbol,
b) Obtener la imagen especular de un árbol (reflejo respecto al eje vertical),
c) Crear tres operaciones que generen una lista con los elementos del árbol recorrido en preorden, inorden y postorden,
d) Comprobar si el árbol está ordenado en inorden, usando para ello únicamente operaciones de árboles (en concreto, no puede utilizarse el apartado anterior).

Solución:

Los apartados a) y c) están resueltos en clase.

Operaciones

Apartado b)

especular: a_bin → a_bin

func especular (a:a_bin) **dev** aie:a_bin
    si vacio?(a) **entonces** aie← Δ
    sino aie←especular(der(a))•raiz(a)•especular(izq(a))
    finsi
finfunc

Apartado d)

mayor_igual: nat a_bin → bool

func mayor_igual(n:natural,a:a_bin) **dev** b:bool
    si vacio?(a) **entonces** b←T
    sino
        b← (n>=(raiz(a)))
            ∧mayor_igual(n, izq(a))
            ∧ mayor_igual(n, der(a))
    finsi
finfunc

menor_igual: nat a_bin → bool

func menor_igual(n:natural,a:a_bin) **dev** b:bool
    si vacio?(a) **entonces** b←T
    sino
        b← (n<(raiz(a)))
            ∧menor_igual(n, izq(a))
            ∧ menor_igual(n, der(a))
    finsi
finfunc

esta_inorden?: a_bin → bool

func esta_inorden? (a:a_bin) **dev** b:bool
    si vacio?(a) **entonces** b←T
    sino b← esta_inorden?(izq(a))
        ∧ (mayor_igual(raíz(a), izq(a)))
        ∧ esta_inorden?(der(a))
        ∧ (menor_igual(raíz(a), der(a)))
    finsi
finfunc

Ejercicio 3.- Se quiere hacer un recorrido de un árbol por niveles (el nivel *k* son todos los nodos que están a distancia *k* de la raíz del árbol). Se pide:

a) nivel_n: a_bin natural → lista, que crea una lista con todos los nodos que se encuentren en el nivel indicado por el *natural* del segundo parámetro;

b) niveles_entre: a_bin natural natural → lista, que crea una lista con todos los nodos que se encuentren entre los niveles indicados por los dos números naturales; y

c) recorrer_niveles: a_bin → lista, que crea una lista formada por todos los niveles del árbol binario.

<u>Solución Operaciones</u>

Apartado a)

nivel_n: árbol→lista

func nivel-n(a:a_bin, n:natural) **dev** l:lista
    si vacio?(a) **entonces** l←[]
    sino **si** n=0 **entonces** l←[raíz(a)]
        sino l←nivel-n (izquierdo(a),n-1)
            ++ nivel-n(derecho(a),n-1)
    finsi
finfunc

Apartado b)

Niveles_entre: árbol→lista

func niveles-entre (a:árbol, n, m:natural)

dev l:lista

si (n<0) V (m<0) V (n>m)

entonces Error (`recorrido incorrecto')

sino si (n=m) **entonces** l←nivel_n(a,n)

sino l←niveles-entre(a, n, m-1)

++nivel-n(a,m)

finsi

finfunc

Apartado c)

recorrer_niveles: árbol→lista

func recorrer_niveles (a:árbol)**dev** l:lista

l←niveles-entre(a, 0, altura(a))

finfunc