Ejercicio 1.- Extender la especificación básica de los árboles binarios con las operaciones para poder hacer:

- `num_nodos: a_bin → natural`, para conseguir el número de nodos en total que hay en un árbol binario;
- `num_hojas: a_bin → natural`, para calcular la cantidad de hojas que tiene un árbol binario;
- `num_padre_de_2: a_bin → a_bin`, que obtiene cúantos nodos tienen dos hijos no vacíos.

Ejercicio 2.- 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".

Solución:

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 3.- *(Examen del Grado en Ingeniería Informática, Enero 2011)* Extender el TAD árboles binarios de naturales (solo hace falta mostrar la especificación de las operaciones básicas de los árboles binarios, pero si se utiliza alguna operación auxiliar hay que presentar su especificación y sus ecuaciones), 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 4.- *(Examen del Grado en Ingeniería Informática, Enero 2012)*
Extender el TAD árboles binarios de naturales (solo hace falta mostrar la especificación de las operaciones básicas de los árboles binarios, pero si se utiliza alguna operación auxiliar hay que presentar su especificación y sus ecuaciones), añadiendo las operaciones siguientes:

- `igual_cantidad?: a_bin a_bin → bool`, detecta si dos árboles binarios tienen la misma cantidad de nodos;
- `igual_forma?: a_bin a_bin → bool`, que comprueba si dos árboles binarios tienen la misma forma;
- `suma_árboles: a_bin a_bin → bool`, que recibe dos árboles binarios de naturales y si tienen la misma forma genera el árbol resultante de sumar los valores de los nodos que ocupan el mismo lugar relativo en cada uno de ellos;

- `cuenta_veces: a_bin a_bin → natural`, que recibe dos árboles binarios y devuelve la cantidad de veces que aparecen los nodos del primero en cualquier posición del segundo.

Ejercicio 5.- 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.

Solución

Operaciones

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
```

Ejercicio 6.- Extender la especificación de los árboles generales vista en clase con las siguientes operaciones:

- `num_nodos: árbol → natural`, para calcular cuántos nodos hay en un árbol general;
- `num_hojas: árbol → natural`, para ver la cantidad total de hojas que tiene un árbol general;
- `max_hijos: árbol → natural`, que obtiene cuál es la mayor cantidad de hijos en un mismo nodo que hay en un árbol general.
- `reflejar: árbol → árbol`, que obtiene la imagen especular de un árbol;
- `frontera: árbol → lista`, que genera una lista formada por los elementos almacenados en las hojas del árbol, tomados de izquierda a derecha.

Solución:

Operaciones

```
Apartado a)
num_nodos: árbol→natural
func  num_nodos (a:árbol) dev n:natural
        n← 1+ num_nodos_b (hijos(a))
finfunc

num_nodos_b: bosque→natural
func  num_nodos_b (b:bosque) dev n:natural

var   prim:arbol
    si vacio?(b) entonces n←0
```

```
    sino     prim←primero(b)
             n←  num_nodos (prim)
                 + num_nodos _b(resto(b))
    fin_si
finfunc
```

Apartado b)

```
num_hojas: árbol→natural
func  num_hojas (a:árbol) dev n:natural
        si vacio?(bosque(a)) entonces n← 1
        sino n←num_hojas_b(hijos(a))
finfunc

num_hojas _b: bosque→natural
func  num_hojas_b (b:bosque) dev n:natural
var   prim:arbol
      si vacio?(b) entonces n←0
      sino
      prim←primero(b)
      n←    num_hojas(prim)+num_hojas_b(resto(b))
      fin_si
finfunc
```

```
Apartado c)
máx_hijos: árbol→natural
func  máx_hijos (a:árbol) dev n:natural
var      num_hijos, max_hijos_b:natural
         num_hijos ← num_hijos(a)
         max_hijos_b← max_hijos_b(bosque(a))
         si (num_hijos > max_hijos_b)
             entonces n← num_hijos
         sino n← máx_hijos _b
         finsi
finfunc

máx_hijos _b: bosque→natural
func  máx_hijos _b (b:bosque) dev n:natural
var   prim:arbol num_hijos_p, max_hijos_r:natural
      si vacio?(b) entonces max_hijos_b←0
        sino prim←primero(b)
        num_hijos_p← num_hijos(prim)
        max_hijos_r← max_hijos_b(resto(b))
        finsi
        si (num_hijos_p > max_hijos_r
            entonces n← num_hijos_p
```

```
        sino n← max_hijos_r
        finsi
finfunc
```

Apartado d)

```
reflejar: árbol→árbol
func reflejar (a:árbol) dev ar:árbol
    ar←raiz(a)•reflejar_b(bosque(a))
finfunc
```

```
bosque_imagen:bosque→bosque

proc bosque_imagen (b, bimag:bosque)
{procedimiento auxiliar que va creando el bosque
imagen de b en bimag}
var prim:árbol
    mientras !vacio(b) hacer prim←primero(b)
                        b←resto(b)
                        bimag←reflejar(prim):bimag
    finmientras
```

```
    finsi

reflejar _b:bosque→bosque
func reflejar_b(b:bosque) dev br:bosque
var bimagen:bosque
    bimagen←[]
    bosque_imagen(b, bimagen)
    {procedimiento auxiliar que va creando el
    bosque imagen de b}
    br←bimag
finfunc

Apartado d)
frontera: árbol→lista
func frontera(a:árbol) dev l:lista
    si num_hijos(a) =0 entonces l←[raíz(a)]
    sino l←frontera_b(hijos(a))
finfunc

func frontera_b (b:bosque) dev l:lista
    si vacio?(b) entonces l←[]
    sino
    l←frontera(primero(b))++frontera_b(resto(b)
    finsi
```

```
finfunc
```

Ejercicio 7.- Extender la especificación de árbol general con las mismas operaciones del **Ejercicio 5**, aunque aplicadas sobre árboles generales en vez de árboles binarios.

Ejercicio 8.- *(Examen del Grado en Ingeniería Informática, Enero 2011)*
Llamaremos a un árbol general de naturales "maestro" si el valor de cada nodo es igual al número de hijos que tiene dicho nodo. Se pide:

a) Especificar completamente el TAD árbol general,

b) Comprobar si un árbol general es "maestro",

c) Buscar el nodo con mayor valor de un árbol maestro (es decir, el que tenga más hijos).

Solución:

operaciones

```
Apartado b)
árbol_maestro?: árbol→ bool
func árbol_maestro(a:árbol) dev b:bool
    b←raiz(a)= num_hijos(a) ∧ bosque_maestro
(bosque(a))
finfunc

func bosque_maestro (b:bosque) dev b:bool
    si vacio?(b) entonces b←T
```

```
    sino b← árbol_maestro(primero(a))
            ∧ bosque_maestro (resto(b))
        finsi
finfunc

Apartado c)
mayor: árbol → natural
func mayor(a:árbol) dev n:natural
    n←maximo(raíz(a), max_bosque(bosque(a))
finfunc

max_bosque: bosque → natural
func max_bosque(b:bosque) dev n:natural
    n←maximo(mayor(primero(b), max_bosque(resto(b))
finfunc
```

Ejercicio 9.- (Examen del Grado en Ingeniería Informática, Enero 2012)

Llamaremos a un árbol general de naturales "creciente" en cada nivel del árbol, la cantidad de nodos que hay en ese nivel es igual al valor del nivel más uno; es decir, el nivel 0 tiene exactamente un nodo, el nivel 1 tiene exactamente dos nodos, el nivel *k* tiene exactamente *k* + 1 nodos. Se pide:

- Especificar completamente el TAD árbol general,
- Comprobar si un árbol general es "creciente",
- Buscar el nodo con mayor cantidad de hijos de un árbol creciente.

Ejercicio 10.- *(Examen de todos los Grados, Enero 2012)* Llamaremos a un árbol general de enteros "progresivo" si cada nodo con hijos tiene a éstos ordenados crecientemente por valor, es decir, si las raíces de cada árbol del bosque aparecen en orden creciente en dicho bosque. Se pide:

- Especificar completamente el TAD árbol general de enteros,
- Comprobar si un árbol general de enteros es "progresivo",
- Transformar un árbol general que no cumpla ser "progresivo" en el árbol "progresivo" equivalente reordenando los hijos en el bosque.