

```
module Exploradores (Explorador , AB(Nil , Bin) , RoseTree(Rose) , foldNat , foldRT ,
    foldAB , expNulo , expId , expHijosRT , expHijosAB , expTail , ifExp , singletons ,
    sufijos , inorder , preorder , postorder , dfsRT , ramasRT , hojasRT ,
    listasQueSuman , listasDeLongitud , (<.>) , (<^>) , (<++>) , (<*>)) where
```

```
import Prelude hiding ((<*>))
```

```
— Definiciones de tipos
```

```
type Explorador a b = a -> [b]
```

```
data AB a = Nil | Bin (AB a) a (AB a) deriving Eq
```

```
data RoseTree a = Rose a [RoseTree a] deriving Eq
```

```
— Definiciones de Show
```

```
instance Show a => Show (RoseTree a) where
    show x = concatMap (++ "\n") (padTree 0 x)
```

```
padTree :: Show a => Int -> RoseTree a -> [String]
padTree i (Rose x ys) = ((pad i) ++ (show x) ) : (concatMap (padTree (i + 4))
    ys)
```

```
pad :: Int -> String
pad i = replicate i ' '
```

```
instance Show a => Show (AB a) where
    show = padAB 0 0
```

```
padAB _ _ Nil = ""
padAB n base (Bin i x d) = pad n ++ show x ++ padAB 4 (base+1) i ++ "\n" ++
    padAB (n+4+base+1) base d where l = length $ show x
```

```
— Ejercicio 1
```

```
expNulo :: Explorador a b
expNulo = (\x -> [])
```

```
expId :: Explorador a a
expId = (\x -> [x])
```

```
expHijosRT :: Explorador (RoseTree a) (RoseTree a)
expHijosRT (Rose raiz hijos) = hijos
```

```
expHijosAB :: Explorador(AB a) (AB a)
expHijosAB Nil = []
expHijosAB (Bin izq r der) = izq:der:[]
```

```
expTail :: Explorador [a] a
expTail = (\a -> if length a /= 0 then tail(a) else [])
```

```
— Ejercicio 2
```

```
foldNat :: (Integer -> b -> b) -> b -> Integer -> b
```

```

foldNat  recu  base  n | n < 0 = error "No se permiten números negativos"
                  | n == 0 = base
                  | otherwise = recu n (foldNat recu base (n-1))

foldRT  :: (a -> [b] -> b) -> RoseTree a -> b
foldRT  recu (Rose root hijos) = recu root (map (foldRT recu) hijos)

foldAB  :: (b -> a -> b -> b) -> b -> AB a -> b
foldAB  recu base Nil = base
foldAB  recu base (Bin izq root der) = recu (foldAB recu base izq) root (foldAB
    recu base der)

—Ejercicio 3
singletons :: Explorador [a] [a]
singletons = foldr (\x recu -> [x]:recu) []

sufijos  :: Explorador [a] [a]
sufijos = foldr (\x recu -> ([x] ++ head(recu)):recu) [[]]

—Ejercicio 4
listasQueSuman :: Explorador Integer [Integer]
listasQueSuman = (\n -> if n == 0 then [[]] else [n]:[y:lista | y <- [1..n-1],
    lista <- listasQueSuman (n-y)])

— El esquema de recursion de foldNat no es adecuado para este ejercicio ya que
  necesitamos hacer en cada paso, n-1 llamados recursivos (desde 1 a n-1).

—Ejercicio 5
preorder :: Explorador (AB a) a
preorder = foldAB (\izq raiz der -> raiz : izq ++ der) []

inorder  :: Explorador (AB a) a
inorder = foldAB (\izq raiz der -> izq ++ [raiz] ++ der) []

postorder :: Explorador (AB a) a
postorder = foldAB (\izq raiz der -> izq ++ der ++ [raiz]) []

—Ejercicio 6
dfsRT :: Explorador (RoseTree a) a
dfsRT = foldRT (\root recu -> root:concat recu)

hojasRT :: Explorador (RoseTree a) a
hojasRT = foldRT (\root recu -> if length recu == 0 then [root] else concat
    recu)

ramasRT :: Explorador (RoseTree a) [a]
ramasRT = foldRT (\root recu -> if length recu == 0 then [[root]] else map ((:)
    root) (concat recu))

—Ejercicio 7
ifExp :: (a->Bool) -> Explorador a b -> Explorador a b -> Explorador a b
ifExp condicion exp1 exp2 = (\estructura -> if condicion estructura then exp1
    estructura else exp2 estructura)

—Ejercicio 8
(<++>) :: Explorador a b -> Explorador a b -> Explorador a b

```

```
(<++>) exp1 exp2 = (\estructura -> exp1 estructura ++ exp2 estructura)
```

—*Ejercicio 9*

```
(<.>) :: Explorador b c -> Explorador a b -> Explorador a c
```

```
(<.>) exp1 exp2 = (\estructura -> concatMap (exp1) (exp2 estructura))
```

—*Ejercicio 10*

```
(<^>) :: Explorador a a -> Integer -> Explorador a a
```

```
(<^>) exp n = (iterate ((<.>) exp) expId) !! fromIntegral (n)
```

—*Ejercicio 11 (implementar al menos una de las dos)*

```
listasDeLongitud :: Explorador Integer [Integer]
```

```
listasDeLongitud = (\n -> [lista | y <- [n..], lista <- listasQueSumanConLong y  
n])
```

```
listasQueSumanConLong :: Integer -> Integer -> [[Integer]]
```

```
listasQueSumanConLong - 0 = [[]]
```

```
listasQueSumanConLong x 1 = [[x]]
```

```
listasQueSumanConLong x n = [y:lista | y <- [1..(x-1)], lista <-  
listasQueSumanConLong (x-y) (n-1)]
```

```
(<*>) :: Explorador a a -> Explorador a [a]
```

```
(<*>) exp = (\estructura -> takeWhile (\elemento -> length elemento /= 0 ) (map  
($estructura) (iterate ((<.>) exp) expId)))
```