

```
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 not $ null a then tail a else [])
```

```
— Ejercicio 2
```

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

```

foldNat recu base 0 = base
foldNat recu base n | n < 0 = error "No se permiten numero negativos"
                  | otherwise = recu (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 (fold izq) root (fold der)
    where fold = foldAB recu base

```

—Ejercicio 3

```

singletons :: Explorador [a] [a]
singletons = map expId

```

```

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

```

— si imaginamos que la lista tiene la forma $[1, 2, \dots, x-1, x, x+1, \dots, n]$, entonces en el caso recursivo del fold, recu tiene todos los sufijos de la lista $[x+1, \dots, n]$. Luego, para completar los sufijos de $[x, x+1, \dots, n]$ solo nos falta agregar aquel que tiene todos los elementos anteriores y x en el orden correcto. Como head recu es el ultimo sufijo construido de la lista $[x+1, \dots, n]$, es exactamente $[x+1, \dots, n]$, por lo que al agregarle x al principio nos queda el sufijo $[x, x+1, \dots, n]$.

—Ejercicio 4

```

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

```

— El esquema de recursion de foldNat no es adecuado para este ejercicio ya que el parametro del llamado recursivo no va descendiendo de a uno. De hecho, dado un n , el Explorador hace n llamados recursivos con valores distintos a la funcion listasQueSuman.

—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 = concatMap exp1 . exp2
```

—Ejercicio 10

```
(<^>) :: Explorador a a -> Integer -> Explorador a a
(<^>) exp = foldNat (<.> exp) expId
```

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

```
listasDeLongitud :: Explorador Integer [Integer]
listasDeLongitud = (\n -> [lista | y <- [(n-1)..],
    lista <- listasConLongQueSuman (n-1) y])
```

```
listasConLongQueSuman :: Integer -> Integer -> [[Integer]]
listasConLongQueSuman = foldNat (\recu -> (\x -> [y:lista | y <- [1..(x-1)],
    lista <- recu (x-y)])) (\x -> [[x]])
```

```
(<*>) :: Explorador a a -> Explorador a [a]
(<*>) exp = (\estructura -> takeWhile (not.null) (map ($estructura)
    (iterate ((<.>) exp) expId)))
```