

# Programação Funcional

## Aula Prática 10

LEI 1º ano (2006/2007)

Considere a seguinte classe para representar *funções parciais finitas*

```
class FF f where
  dom :: (Ord a) => (f a b) -> [a]
  procura :: (Ord a) => (f a b) -> a -> Maybe b
  acrescenta :: (Ord a) => (f a b) -> a -> b -> f a b
  remove :: (Ord a) => (f a b) -> a -> f a b
  lista :: (Ord a) => (f a b) -> [(a,b)]
```

Considere agora as seguintes alternativas para implementar funções finitas:

### Listas ordenadas de pares

```
data (Ord a) => LOrd a b = L [(a,b)]
```

### Árvores binárias de procura

```
data (Ord a) => ABP a b = V | N (a,b) (ABP a b) (ABP a b)
```

### Tarefas

1. Considere a seguinte definição de instância.

```
instance FF LOrd where
  dom (L l) = map fst l
  procura (L l) x = case (takeWhile ((==x).fst) (dropWhile ((<x).fst) l)) of
    ((_,b):_) -> Just b
  acrescenta (L l) x y = let (a,b) = break ((<x) .fst) l
    in L (a ++ ((x,y):(dropWhile ((==x).fst) b)))
  remove (L l) x = let (a,b) = break ((<x) .fst) l
    in L (a ++ (dropWhile ((==x).fst) b))
  lista (L l) = l
```

Reescreva a definição acima sem usar funções de ordem superior (`map`, `filter` e `takeWhile`)

2. Defina `ABP` como uma instância da classe `FF`.
3. Assumindo que ambos os tipos `a` e `b` são instâncias de `Show`, defina `LOrd a b` e `ABP a b` como instâncias de `Show`. Faça-o de forma que a informação seja listada por ordem crescente da chave, e um registo por linha.