Parte II

1. Considere expressões representadas como listas de números inteiros contêm apenas multiplicações, adições ou subtrações que associam à direita. Por exemplo, `[5,3,8]`— pode corresponder a `5 + (3 + 8)`, a `5 - (3 + 8)`, a `5 * (3 - 8)`, etc. Defina a função `resultado :: [Int] -> Maybe String` que recebe a lista e um valor e que determina se é possível, utilizando apenas multiplicações, adições e subtrações, obter o valor fornecido a partir de todos os números da lista.

   Por exemplo, `resultado [5,3,8] 55` deve produzir `Just "5 * (3 + 8)"`, mas `resultado [10,2] 6` deve produzir `Nothing`, pois com os valores apenas conseguimos obter os resultados `10 + 2 == 12`, `10 - 2 == 10` e `10 + 2 == 20`, todos diferentes de `6`.

2. Considere a seguinte definição da função `deleteMin` que remove todas as ocorrências do menor valor de uma lista:

```
deleteMin :: Ord a => [a] -> [a]
deleteMin l =  let  m = minimum l
               in   filter (/= m) l
```

   a) Relembre a técnica de *tupling* e comece por definir uma função auxiliar `minRem :: Ord a => [a] -> (a,[a])` que calcula o mínimo de uma lista e a lista sem esse mínimo.

   Use a função `minRem` para obter uma definição alternativa da função `deleteMin` que percorre uma única vez a lista de entrada.

   b) Use a função `minRem` para definir uma função que ordena uma lista removendo as repetições de elementos.

3. Considere a seguinte representação de árvores binárias em que os valores estão nas suas folhas. À direita mostra-se ainda um exemplo de uma destas árvores.

```
data LTree a = Leaf a
             | Fork (LTree a) (LTree a)
```

```
lt1 :: LTree Char
lt1 = Fork (Leaf 'A')
           (Fork (Fork (Leaf 'B')
                       (Leaf 'C'))
                 (Leaf 'D'))
```

   a) Cada folha de uma destas árvores pode ser identificada pelo caminho (*path*) que a liga à raíz da árvore. Por exemplo, na árvore `lt1` acima, o caminho da raiz até à folha `'C'` é DIREITA,ESQUERDA,DIREITA. Se em vez disso representarmos estas *decisões* como `'0'` e `'1'`, esse caminho pode ser descrito pela *String* `"101"`

   Defina a função `selectLeaf :: String -> LTree a -> Maybe a` que determina a folha de uma árvore identificada por um dado caminho. Se esse caminho não conduzir a uma folha, a função deve retornar `Nothing`.

   b) Considere agora a função apresentada ao lado, que, dada uma árvore destas calcula a lista de todas as suas folhas bem como o *caminho* correspondente. Por exemplo, `elemPath lt1 = [("0",'A'),("100",'B'),("101",'C'),("11",'D')]`

```
elemPath :: LTree a -> [(String,a)]
elemPath (Leaf x) = [("",x)]
elemPath (Fork e d)
    =    [('0':p,x) | (p,x) <- elemPath e]
      ++ [('1':p,x) | (p,x) <- elemPath d]
```

   Defina a função `buildLTree :: [(String,a)] -> LTree a` inversa da anterior, no sentido em que, para qualquer árvore `a` se verifica que `builLTree (elemPath a) == a`.