Exame

Programação Funcional
1° Ano – LEI/LCC

17 de Fevereiro de 2012
Duração: 2 horas

Parte I

Esta parte do teste representa 12 valores da cotação total. Cada alínea está cotada em 2 valores. **A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.**

1. Defina uma função `posNeg :: [Int] -> (Int,Int)` que, dada uma lista de inteiros, conta o número de valores positivos e o número de valores negativos, e devolve um par com essa informação. Certifique-se que a função que definiu **percorre a lista apenas uma vez**.

2. Defina a função `tiraPrefixo :: String -> String -> Maybe String` que, caso a primeira string seja prefixo da segunda, retira-lhe esse prefixo. Por exemplo: `tiraPrefixo ‘‘mal’’ ‘‘malcriado’’ = Just ‘‘criado’’`, e `tiraPrefixo ‘‘mal’’ ‘‘amalgama’’ = Nothing`.

3. Apresente uma definição alternativa da função `fun` usando recursividade explicita (i.e. sem usar as funções `map` e `filter`)

   ```
   fun f l = product (map f (filter (>0) l))
   ```

4. Considere o seguinte tipo para representar árvores binárias:

   ```
   data ArvBin a = Vazia | Nodo a (ArvBin a) (ArvBin a)
   ```

   Defina a função `insere :: Ord a => a -> ArvBin a -> ArvBin a`, que insere um elemento numa árvore binária de procura.

5. Para representar a informação de uma prova de atletismo, definiram-se os seguintes tipos de dados:

   ```
   type Concorrentes = [(Num, String)]      -- número e nome
   type Num = Int
   type Prova = [(Num, Float)]              -- número e tempo gasto na prova
   ```

   (a) Defina a função `nomes :: Prova -> Concorrentes -> [(String,Float)]`, que dá a informação da prova, associando ao nome de cada concorrente o seu tempo.

   (b) Defina a função `ordena :: Prova -> Prova`, que ordena os concorrentes por ordem de chegada.

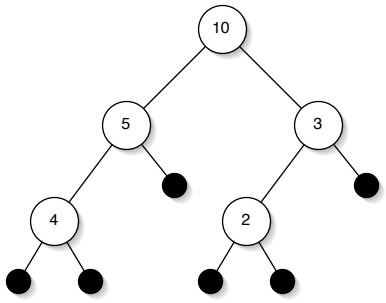
Parte II

1. Considere o seguinte tipo para representar árvores binárias:

   ```
   data ArvBin a = Vazia | Nodo a (ArvBin a) (ArvBin a)
   ```

   A cada elemento de uma árvore pode ser associado o seu *"caminho"*, que não é mais do que uma lista de Booleanos. Por exemplo, na árvore da figura temos a seguinte correspondência entre nodos e caminhos:

   - O caminho `[True,False]` corresponde ao nodo 2.
   - O caminho `[False,False]` corresponde ao nodo 4.
   - O caminho `[]` corresponde ao nodo 10 (a raíz da árvore).
   - O caminho `[True,True]` não corresponde a nenhum nodo da árvore.

   (a) Defina uma função `camValido :: ArvBin a -> [Bool] -> Bool`, que dada uma árvore e um caminho, diz se esse caminho é válido (i.e., se corresponde a algum nodo da árvore)

   (b) Defina a função `caminho :: (Eq a) => ArvBin a -> a -> Maybe [Bool]`, que dada uma árvore e um elemento. calcula um caminho desde a raíz até esse elemento. A função deverá retornar `Nothing` se o elemento não pertencer à árvore.

2. O problema das N raínhas consiste em colocar N rainhas num tabuleiro de xadrez com N linhas e N colunas, de tal forma que nenhuma rainha está ameaçada por outra.
   Note que uma rainha ameaça todas as posições que estão na mesma linha, na mesma coluna ou nas mesmas diagonais. No tabuleiro ao lado apresenta-se uma solução para este problema quando N é 7.

		Q				
						Q
			Q			
Q						
				Q		
	Q					
					Q	

   Uma forma de representar estas soluções é usando listas de strings. Por exemplo a lista `["Q_", "_Q"]` representa um tabuleiro de dimensão 2 com duas rainhas colocadas nos cantos superior esquerdo e inferior direito. Para o efeito fez-se a seguinte declaração de tipo

   ```
   type Tabuleiro = [String]
   ```

   (a) Defina uma função `rainhasOK` que testa se uma dada solução está correcta. Por exemplo, para o tabuleiro `["Q_", "_Q"]` a função deve dar `False` como resultado, uma vez que as raínhas se encontram na mesma diagonal.

   (b) Defina a função `solucoes :: Int -> [Tabuleiro]` qua calcula todas as soluções possíveis para o problema com um dado número de raínhas. Por exemplo, para 4 raínhas a soluções serão

   ```
   [["_Q__","___Q","Q___","__Q_"],["__Q_","Q___","___Q","_Q__"]]
   ```