Teste

Programação Funcional
1° Ano – LEI/LCC

21 de Janeiro de 2012
Duração: 2 horas (teste completo) / 1 hora (Parte II)

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 `maxSumPair :: [(Int,Int)] -> (Int,Int)` que, dada uma lista não vazia de pares de inteiros, calcula um par cuja soma é máxima. Por exemplo, `maxSumPairs [(1,10), (6,6), (10,1)]` tem valor `(6,6)`.

2. Defina uma função `maxes :: [(Int,Int)] -> (Int,Int)` que, dada uma lista não vazia de pares de inteiros, calcula um par em que a primeira componente é a maior das primeiras componentes e a segunda componente é a maior das segundas componentes. Por exemplo, `maxes [(1,10), (6,6), (10,1)]` tem valor `(10,10)`. Certifique-se que a função que definiu percorre a lista **apenas uma vez**.

3. Considere o seguinte tipo para representar árvores binárias:

   ```
   data BTree a = Empty | Node a (BTree a) (BTree a)
   ```

   Defina uma função `procura` que testa se um elemento pertence a uma árvore binária (não necessariamente de procura).

   Apresente ainda o tipo da função.

4. A função `concatMap :: (a -> [b]) -> [a] -> [b]` pode ser definida como

   ```
   concatMap f l = concat (map f l)
   ```

   Apresente uma definição alternativa desta função (`concatMap`) sem usar a função `map` (i.e., usando recursividade explícita).

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 `junta :: Prova -> Concorrentes -> [(Num,String,Float)]`, que junte a informação das duas tabelas de informação.

   (b) Considere a seguinte função:

   ```
   quantos :: Float -> Prova -> Int
   quantos x p = length (filter (aux x) p)
   ```

   Defina a função `aux` de forma a que a função `quantos` indique quantos atletas fizeram tempos abaixo de um dado valor `x`. Declare também o tipo da função `aux`.

Parte II

Para controlar um veículo tele-comandado, dispõe-se dos seguintes comandos:

RD – Rodar à direita: roda o veículo 90º no sentido dos ponteiros do relógio;

RE – Rodar à esquerda: roda o veículo 90º no sentido contrário ao dos ponteiros do relógio;

AV – Avançar: avança o veículo uma posição.

Para representar esses comandos definiu-se o tipo:

```
data Cmd = RD | RE | AV
     deriving (Eq,Show)
```

1. Defina:
   - O tipo `Pos` que represente a posição do veículo (i.e. coordenadas do ponto e orientação).
   - A função `next :: Pos -> Cmd -> Pos`, que dada uma posição inícial e um comando calcula a posição do veículo após executar esse comando.
2. Defina a função `percurso` que, dada uma posição inicial e uma lista de comandos, retorne a lista das posições percorridas pelo veículo (o percurso realizado).
3. Defina um predicado que verifique, dadas as posições iniciais de dois veículos e as respectivas listas de comandos (uma por veículo), determine se ocorre uma colisão entre esses veículos (i.e. se existe um momento onde ambos os veículos ocupam uma posição no plano).

   Considere para o efeito que após realizados todos os comandos contidos na lista, o veículo permanece "estacionado" na posição final.
4. Pretende-se definir um simulador (`simula :: IO()`) para um veículo com a funcionalidade descrita nas alíneas anteriores. Para tal, considere que:
   - o veículo desloca-se num tabuleiro de dimensões 10x10 (coordenadas (0,0) até (9,9));
   - inicialmente são colocados obstáculos em 10 posições aleatórias do tabuleiro;
   - a posição inicial do veículo é (0,0) com orientação vertical (virado para norte);
   - após a inicialização do tabuleiro, o simulador entra num ciclo de interação com o utilizador solicitando um comando, e apresentando a nova posição/orientação do veículo;
   - a interacção termina quando uma das seguintes condições se verificar:
     - o veículo sair fora dos limites do tabuleiro;
     - o veículo chocar com um dos obstáculos colocados no tabuleiro;
     - o veículo voltar a uma posição em que já tenha estado.

   Relembre que para gerar um número aleatório compreendido entre `x` e `y` pode utilizar a função `randomRIO::(a,a) -> IO a`.