Ficha 4

Programação Funcional

2015/16

1. Considere as seguintes definições.

```
type Ponto = (Float,Float)            -- (abcissa,ordenada)
type Rectangulo = (Ponto,Float,Float) -- (canto sup.esq., larg, alt)
type Triangulo = (Ponto,Ponto,Ponto)
type Poligonal = [Ponto]

distancia :: Ponto -> Ponto -> Float
distancia (a,b) (c,d) = sqrt (((c-a)^2) + ((b-d)^2))
```

   (a) Defina uma função que calcule o comprimento de uma linha poligonal.

   (b) Defina uma função que converta um elemento do tipo `Triangulo` na correspondente linha poligonal.

   (c) Repita o alínea anterior para elementos do tipo `Rectangulo`.

   (d) Defina uma função `fechada` que testa se uma dada linha poligonal é ou não fechada.

   (e) Defina uma função `triangula` que, dada uma linha poligonal fechada e convexa, calcule uma lista de triângulos cuja soma das áreas seja igual à área delimitada pela linha poligonal.

   (f) Suponha que existe uma função `areaTriangulo` que calcula a área de um triângulo.

```
areaTriangulo (x,y,z)
   = let a = distancia x y
         b = distancia y z
         c = distancia z x
         s = (a+b+c) / 2 -- semi-perimetro
     in -- formula de Heron
        sqrt (s*(s-a)*(s-b)*(s-c))
```

   Usando essa função, defina uma função que calcule a área delimitada por uma linha poligonal fechada e convexa.

   (g) Defina uma função `mover` que, dada uma linha poligonal e um ponto, dá como resultado uma linha poligonal idêntica à primeira mas tendo como ponto inicial o ponto dado. Por exemplo, ao mover o triângulo `[(1,1),(10,10),(10,1),(1,1)]` para o ponto (1,2) devemos obter o triângulo `[(1,2),(10,11),(10,2),(1,2)]`.

   (h) Defina uma função `zoom2` que, dada uma linha poligonal, dê como resultado uma linha poligonal semelhante e com o mesmo ponto inicial mas em que o comprimento de cada segmento de recta é multiplicado por 2. Por exemplo, o rectângulo

`[(1,1),(1,3),(4,3),(4,1),(1,1)]`

   deverá ser transformado em `[(1,1),(1,5),(7,5),(7,1),(1,1)]`

2. Considere as seguintes definições de tipos para representar uma tabela de registo de temperaturas.

```
type TabTemp = [(Data,Temp,Temp)]  -- (data, temp. mínima, temp. máxima)
type Data = (Int,Int,Int)          -- (ano, mês, dia)
type Temp = Float
```

   (a) Defina a função `médias :: TabTemp -> [(Data,Temp)]` que constroi a lista com as temperaturas médias de cada dia.

   (b) Defina a função `decrescente :: TabTemp -> Bool` que testa se a tabela está ordenada por ordem decrescente de data. (Nota: pode usar o operador $>$ para comparar directamente duas datas.)

   (c) Defina a função `conta :: [Data] -> TabTemp -> Int` que, dada uma lista de datas e a tabela de registo de temperaturas, conta quantas das datas da lista têm registo de na tabela.

3. Um multi-conjunto é um conjunto que admite elementos repetidos. É diferente de uma lista porque a ordem dos elementos não é relevante. Uma forma de implementar multi-conjuntos em Haskell é através de uma lista de pares, onde cada par regista um elemento e o respectivo número de ocorrências:

   ```
   type MSet a = [(a,Int)]
   ```

   Uma lista que representa um multi-conjunto não deve ter mais do que um par a contabilizar o número de ocorrências de um elemento, e o número de ocorrências deve ser sempre estritamente positivo. O multi-conjunto de caracteres `{'b','a','c','a','b','a'}` poderia, por exemplo, ser representado pela lista `[('b',2),('a',3),('c',1)]`.

   (a) Defina a função `union :: Eq a => MSet a -> MSet a -> MSet a` que calcula a união de dois multi-conjuntos. Por exemplo,

   ```
   > union [('a',3),('b',2),('c',1)] [('d',5),('b',1)]
   [('a',3),('b',3),('c',1),('d',5)]
   ```

   (b) Defina a função `intersect :: Eq a => MSet a -> MSet a -> MSet a` que calcula a intersecção de dois multi-conjuntos. Por exemplo,

   ```
   > intersect [('a',3),('b',5),('c',1)] [('d',5),('b',2)]
   [('b',2)]
   ```

   (c) Defina a função `diff :: Eq a => MSet a -> MSet a -> MSet a` que calcula a diferença de dois multi-conjuntos. Por exemplo,

   ```
   > diff [('a',3),('b',5),('c',1)] [('d',5),('b',2)]
   [('a',3),('b',3)],('c',1)]
   ```

   (d) Defina a função `ordena :: MSet a -> MSet a` que ordena um multi-conjunto pelo número crescente de ocorrências. Por exemplo,

   ```
   > ordena [('b',2),('a',3),('c',1)]
   [('c',1),('b',2),('a',3)]
   ```

   (e) Defina a função `moda :: MSet a -> [a]` que devolve a lista dos elementos com maior número de ocorrências. Por exemplo,

   ```
   > moda [('b',2),('a',3),('c',1),('d',3)]
   ['a','d']
   ```