Aula Teórico-prática 4

Programação Funcional

LEI 1º ano

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 exercício 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 que a informação sobre um stock de uma loja está armazenada numa lista de tuplos (com o nome do produto, o seu preço unitário, e a quantidade em stock desse produto) de acordo com as seguintes declarações de tipos:

```
type Stock = [(Produto,Preco,Quantidade)]
type Produto = String
type Preco = Float
type Quantidade = Float
```

Assuma que um produto não ocorre mais do que uma vez na lista de stock.

(a) Defina as seguintes funções de manipulação e consulta do stock:

i. `quantos::Stock->Int`, que indica quantos produtos existem em stock.

ii. `emStock::Produto->Stock->Quantidade`, que indica a quantidade de um dado produto existente em stock.

iii. `consulta::Produto->Stock->(Preco,Quantidade)`, que indica o preço e a quantidade de um dado produto existente em stock.

iv. `tabPrecos::Stock->[(Produto,Preco)]`, para construir uma tabela de preços.

v. `valorTotal::Stock->Float`, para calcular o valor total do stock.

vi. `inflacao::Float->Stock->Stock`, que aumenta uma dada percentagem a todos os preços.

vii. `omaisBarato::Stock->(Produto,Preco)`, que indica o produto mais barato e o seu preço.

viii. `maisCaros::Preco->Stock->[Produto]`, que constroi a lista dos produtos caros (i.e., acima de um dado preço).

(b) Considere agora que tem a seguinte declaração de tipo para modelar uma lista de compras:

```
type ListaCompras = [(Produto,Quantidade)]
```

Defina as funções que se seguem:

i. `verifLista::ListaCompras->Stock->Bool`, que verifica se todos os pedidos podem ser satisfeitos.

ii. `falhas::ListaCompras->Stock->ListaCompras`, que constroi a lista dos pedidos não satisfeitos.

iii. `custoTotal::ListaCompras->Stock->Float`, que calcula o custo total da lista de compras.

iv. `partePreco::Preco->ListaCompras->Stock->(ListaCompras,ListaCompras)`, que parte a lista de compras em duas: uma lista com os items inferiores a um dado preço, e a outra com os items superiores ou iguais a esse preço.