Ficha 1

Programação Funcional

1º ano

1. Recorde as seguintes funções pré-definidas do Haskell:

   - `length l`: o número de elementos da lista `l`
   - `head l`: a cabeça da lista (não vazia) `l`
   - `tail l`: a cauda lista (não vazia) `l`
   - `last l`: o último elemento da lista (não vazia) `l`
   - `sqrt x`: a raiz quadrada de `x`
   - `div x y`: a divisão inteira de `x` por `y`
   - `mod x y`: o resto da divisão inteira de `x` por `y`

   Defina as seguintes funções:

   (a) `perimetro` – que calcula o perímetro de uma circunferência, dado o comprimento do seu raio.

   (b) `dist` – que calcula a distância entre dois pontos no plano Cartesiano. Cada ponto é um par de valores do tipo `Float`.

   (c) `primUlt` – que recebe uma lista e devolve um par com o primeiro e o último elemento dessa lista.

   (d) `multiplo` – tal que `multiplo m n` testa se o número inteiro `m` é múltiplo de `n`.

   (e) `truncaImpar` – que recebe uma lista e, se o comprimento da lista for ímpar retira-lhe o primeiro elemento, caso contrário devolve a própria lista.

   (f) `max2` – que calcula o maior de dois números inteiros.

   (g) `max3` – que calcula o maior de três números inteiros. Para isso apresente duas definições alternativas: recorrendo ou não à função `max2` definida na alínea anterior.

2. Num triângulo verifica-se sempre que a soma dos comprimentos de dois dos lados é superior à do terceiro. A esta propriedade chama-se *desigualdade triangular*. Defina uma função que, dados três números, teste se esses números correspondem aos comprimentos dos lados de um triângulo.

3. Vamos representar um ponto por um par de números que representam as suas coordenadas no plano Cartesiano.

   ```
   type Ponto = (Float,Float)
   ```

   (a) Defina uma função que recebe 3 pontos que são os vértices de um triângulo e devolve um tuplo com o comprimento dos seus lados.

   (b) Defina uma função que recebe 3 pontos que são os vértices de um triângulo e calcula o perímetro desse triângulo.

   (c) Defina uma função que recebe 2 pontos que são os vértices da diagonal de um rectângulo paralelo aos eixos e constroi uma lista com os 4 pontos desse rectângulo.

4. Defina uma função que recebe os (3) coeficientes de um polinómio de 2º grau e que calcula o número de raízes (reais) desse polinómio.

5. Usando a função anterior, defina uma função que, dados os coeficientes de um polinómio de 2º grau, calcula a lista das suas raízes reais.

6. As funções das duas alíneas anteriores podem receber um tuplo com os coeficientes do polinómio, ou receber os 3 coeficientes separadamente. Defina a versão alternativa ao que definiu acima.

7. Utilisando as funções `ord::Char->Int` e `chr::Int->Char` defina as seguintes funções:

   (a) `isLower :: Char -> Bool`
   (b) `isDigit :: Char -> Bool`
   (c) `isAlpha :: Char -> Bool`
   (d) `toUpper :: Char -> Char`
   (e) `intToDigit :: Int -> Char`
   (f) `digitToInt :: Char -> Int`

   Obs: todas estas funções já estão definidas no módulo `Char` (ou `Data.Char`).

8. Vamos representar horas por um par de números inteiros:

   ```
   type Hora = (Int,Int)
   ```

   Assim o par `(0,15)` significa *meia noite e um quarto* e `(13,45)` *duas menos um quarto.* Defina funções para:

   (a) testar se um par de inteiros representa uma hora do dia válida;
   (b) testar se uma hora é ou não depois de outra (comparação);
   (c) converter um valor em horas (par de inteiros) para minutos (inteiro);
   (d) converter um valor em minutos para horas;
   (e) calcular a diferença entre duas horas (cujo resultado deve ser o número de minutos)
   (f) adicionar um determinado número de minutos a uma dada hora.

9. Analise a seguinte definição e apresente uma definição alternativa que use concordância de padrões em vez dos *if*s.

   ```
   opp :: (Int,(Int,Int)) -> Int
   opp z = if ((fst z) == 1)
           then (fst (snd z)) + (snd (snd z))
           else if ((fst z) == 2)
                then (fst (snd z)) - (snd (snd z))
                else 0
   ```

10. Defina recursivamente as seguintes funções sobre listas:

    (a) `dobros :: [Float] -> [Float]` que recebe uma lista e produz a lista em que cada elemento é o dobro do valor correspondente na lista de entrada.
    (b) `ocorre :: Char -> String -> Int` que calcula o número de vezes que um caracter ocorre numa string.
    (c) `pmaior :: Int -> [Int] -> Int` que recebe um inteiro n e uma lista l de inteiros e devolve o primeiro número em l que é maior do que n. Se nenhum número em l for maior do que n, devolve n.
    (d) `repetidos :: [Int] -> Bool` que testa se uma lista tem elementos repetidos.
    (e) `nums :: String -> [Int]` recebe uma string e devolve uma lista com os algarismos que occorem nessa string, pela mesma ordem. (Obs: relembre as funções da Ficha 2).
    (f) `tresUlt :: [a] -> [a]` devolve os últimos três elementos de uma lista, Se a lista de entrada tiver menos de três elementos, devolve a própria lista.

(g) `posImpares :: [a] -> [a]` calcula a lista com os elementos que occorem nas posiçãoes impares da lista de entrada.

11. Considere o seguinte tipo de dados para armazenar informação sobre uma turma de alunos:

```
type Aluno = (Numero,Nome,ParteI,ParteII)
type Numero = Int
type Nome = String
type ParteI = Float
type ParteII = Float
type Turma = [Aluno]
```

Defina funções para:

(a) Testar se uma turma é válida (i.e., os alunos tem todos números diferentes, as notas da Parte I estão entre 0 e 12, e as notas da Parte II entre 0 e 8).

(b) Seleciona os alunos que passaram (i.e., a nota da Parte I não é inferior a 8, e a soma das nota da Parte I e II é superior ou igual a 9,5).

(c) Calcula a nota final dos alunos que passaram.

(d) Calcular a média das notas dos alunos que passaram.

(e) Determinar o nome do aluno com nota mais alta.

12. Assumindo que uma hora é representada por um par de inteiros, uma viagem pode ser representada por uma sequência de etapas, onde cada etapa é representada por um par de horas (partida, chegada):

```
type Hora = (Int,Int)
type Etapa = (Hora,Hora)
type Viagem = [Etapa]
```

Por exemplo, se uma viagem for

```
[((9,30), (10,25)), ((11,20), (12,45)) , ((13,30), (14,45))]
```

significa que teve três etapas:

- a primeira começou às 9 e um quarto e terminou às 10 e 25;
- a segunda começou às 11 e 20 e terminou à uma menos um quarto;
- a terceira começou às 1 e meia e terminou às 3 menos um quarto;

Utilizando as funções sobre horas que definiu na alínea 8, defina as seguintes funções:

(a) Testar se uma etapa está bem construída (i.e., o tempo de chegada é superior ao de partida e as horas são válidas).

(b) Testa se uma viagem está bem construída (i.e., se para cada etapa, o tempo de chegada é superior ao de partida, e que a etapa seguinte começa depois de a etapa anterior ter terminado).

(c) Calcular a hora de partida e de chegada de uma dada viagem.

(d) Dada uma viagem válida, calcular o tempo total de viagem efectiva.

(e) Calcular o tempo total de espera.

(f) Calcular o tempo total da viagem (a soma dos tempos de espera e de viagem efectiva).