Programação Funcional

Ficha 1

Funções não recursivas

1. Usando 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 e os respectivos tipos:

   (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 `Double`.

   (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, usando a função `max2`.

2. Defina as seguintes funções sobre polinómios de 2ª grau:

   (a) A função `nRaizes` que recebe os (3) coeficientes de um polinómio de 2º grau e que calcula o número de raízes (reais) desse polinómio.

   (b) A função `raizes` que, usando a função anterior, recebe os coeficientes do polinómio e calcula a lista das suas raízes reais.

3. 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.

4. Repita o exercício anterior assumindo agora que as horas são representadas por um novo tipo de dados:

```
data Hora = H Int Int deriving (Show,Eq)
```

Com este novo tipo a hora *meia noite e um quarto* é representada por `H 0 15` e a hora *duas menos um quarto* por `H 13 45`.

5. Considere o seguinte tipo de dados para representar os possíveis estados de um semáforo:

```
data Semaforo = Verde | Amarelo | Vermelho deriving (Show,Eq)
```

(a) Defina a função `next :: Semaforo -> Semaforo` que calcula o próximo estado de um semáforo.

(b) Defina a função `stop :: Semaforo -> Bool` que determina se é obrigatório parar num semáforo.

(c) Defina a função `safe :: Semaforo -> Semaforo -> Bool` que testa se o estado de dois semáforos num cruzamento é seguro.

6. Um ponto num plano pode ser representado por um sistema de coordenadas Cartesiano (distâncias aos eixos vertical e horizontal) ou por um sistema de coordenadas Polar (distância à origem e ângulo do respectivo vector com o eixo horizontal).

```
data Ponto = Cartesiano Double Double | Polar Double Double
             deriving (Show,Eq)
```

Com este tipo o ponto `Cartesiano (-1) 0` pode alternativamente ser representado por `Polar 1 pi`. Defina as seguintes funções:

(a) `posx :: Ponto -> Double` que calcula a distância de um ponto ao eixo vertical.

(b) `posy :: Ponto -> Double` que calcula a distância de um ponto ao eixo horizontal.

(c) `raio :: Ponto -> Double` que calcula a distância de um ponto à origem.

(d) `angulo :: Ponto -> Double` que calcula o ângulo entre o vector que liga a origem a um ponto e o eixo horizontal.

(e) `dist :: Ponto -> Ponto -> Double` que calcula a distância entre dois pontos.

7. Considere o seguinte tipo de dados para representar figuras geométricas num plano.

```
data Figura = Circulo Ponto Double
            | Rectangulo Ponto Ponto
            | Triangulo Ponto Ponto Ponto
              deriving (Show,Eq)
```

Uma figura pode ser um círculo centrado um determinado ponto e com um determinado raio, um rectângulo paralelo aos eixos representado por dois pontos que são vértices da sua diagonal, ou um triângulo representado pelos três pontos dos seus vértices. Defina as seguintes funções:

(a) Defina a função `poligono :: Figura -> Bool` que testa se uma figura é um polígono.

(b) Defina a função `vertices :: Figura -> [Ponto]` que calcula a lista dos vertices de uma figura.

(c) Complete a seguinte definição cujo objectivo é calcular a área de uma figura:

```
area :: Figura -> Double
area (Triangulo p1 p2 p3) =
       let a = dist p1 p2
           b = dist p2 p3
           c = dist p3 p1
           s = (a+b+c) / 2 -- semi-perimetro
       in sqrt (s*(s-a)*(s-b)*(s-c)) -- formula de Heron
...
```

(d) Defina a função `perimetro :: Figura -> Double` que calcula o perímetro de uma figura.

8. Utilizando as funções `ord :: Char -> Int` e `chr :: Int -> Char` do módulo `Data.Char`, defina as seguintes funções:

   (a) `isLower :: Char -> Bool`, que testa se um `Char` é uma minúscula.

   (b) `isDigit :: Char -> Bool`, que testa se um `Char` é um dígito.

   (c) `isAlpha :: Char -> Bool`, que testa se um `Char` é uma letra.

   (d) `toUpper :: Char -> Char`, que converte uma letra para a respectiva maiúscula.

   (e) `intToDigit :: Int -> Char`, que converte um número entre `0` e `9` para o respectivo dígito.

   (f) `digitToInt :: Char -> Int`, que converte um dígito para o respectivo inteiro.

   Note que todas estas funções já estão também pré-definidas nesse módulo.