Programação Funcional

Ficha 3

Gestão de informação em listas

1. 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):

```
data Hora = H Int Int
          deriving Show

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

   Por exemplo, se uma viagem for

```
[(H 9 30, H 10 25), (H 11 20, H 12 45), (H 13 30, H 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;

   Para este problema, vamos trabalhar apenas com viagens que começam e acabam no mesmo dia. Utilizando as funções sobre horas que definiu na Ficha 1, 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 se a etapa seguinte começa depois da 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).

2. Considere as seguinte definição de um tipo para representar linhas poligonais.

```
type Poligonal = [Ponto]
```

   O tipo `Ponto` é idêntico ao definido na Ficha 1. Nas resolução das alíneas seguintes pode utilizar funções definidas nessa ficha.

   (a) Defina a função para calcular o comprimento de uma linha poligonal.

   (b) Defina uma função para testar se uma dada linha poligonal é ou não fechada.

(c) Defina a função `triangula :: Poligonal -> [Figura]` 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. O tipo `Figura` é idêntico ao definido na Ficha 1

(d) Defina uma função para calcular a área delimitada por uma linha poligonal fechada e convexa.

(e) Defina a função `mover :: Poligonal -> Ponto -> Poligonal` 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.

(f) Defina a função `zoom :: Double -> Poligonal -> Poligonal` que, dada um factor de escala e 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 pelo factor dado.

3. Para armazenar uma agenda de contactos telefónicos e de correio electrónico definiram-se os seguintes tipos de dados. Não existem nomes repetidos na agenda e para cada nome existe uma lista de contactos.

```
data Contacto = Casa Integer
              | Trab Integer
              | Tlm Integer
              | Email String
              deriving Show

type Nome = String
type Agenda = [(Nome, [Contacto])]
```

(a) Defina a função `acrescEmail :: Nome -> String -> Agenda -> Agenda` que, dado um nome, um email e uma agenda, acrescenta essa informação à agenda.

(b) Defina a função `verEmails :: Nome -> Agenda -> Maybe [String]` que, dado um nome e uma agenda, retorna a lista dos emails associados a esse nome. Se esse nome não existir na agenda a função deve retornar `Nothing`.

(c) Defina a função `consTelefs :: [Contacto] -> [Integer]` que, dada uma lista de contactos, retorna a lista de todos os números de telefone dessa lista (tanto telefones fixos como telemóveis).

(d) Defina a função `casa :: Nome -> Agenda -> Maybe Integer` que, dado um nome e uma agenda, retorna o número de telefone de casa (caso exista).

4. Pretende-se guardar informação sobre os aniversários das pessoas numa tabela que associa o nome de cada pessoa à sua data de nascimento. Para isso, declarou-se a seguinte estrutura de dados

```
type Dia = Int
type Mes = Int
type Ano = Int
type Nome = String

data Data = D Dia Mes Ano
          deriving Show

type TabDN = [(Nome,Data)]
```

(a) Defina a função `procura :: Nome -> TabDN -> Maybe Data`, que indica a data de nascimento de uma dada pessoa, caso o seu nome exista na tabela.

(b) Defina a função `idade :: Data -> Nome -> TabDN -> Maybe Int`, que calcula a idade de uma pessoa numa dada data.

(c) Defina a função `anterior :: Data -> Data -> Bool`, que testa se uma data é anterior a outra data.

(d) Defina a função `ordena :: TabDN -> TabDN`, que ordena uma tabela de datas de nascimento, por ordem crescente das datas de nascimento.

(e) Defina a função `porIdade:: Data -> TabDN -> [(Nome,Int)]`, que apresenta o nome e a idade das pessoas, numa dada data, por ordem crescente da idade das pessoas.

5. Considere o seguinte tipo de dados que descreve a informação de um extracto bancário. Cada valor deste tipo indica o saldo inicial e uma lista de movimentos. Cada movimento é representado por um triplo que indica a data da operação, a sua descrição e a quantia movimentada (em que os valores são sempre números positivos).

```
data Movimento = Credito Float | Debito Float
               deriving Show

data Data = D Int Int Int
          deriving Show

data Extracto = Ext Float [(Data, String, Movimento)]
              deriving Show
```

(a) Construa a função `extValor :: Extracto -> Float -> [Movimento]` que produz uma lista de todos os movimentos (créditos ou débitos) superiores a um determinado valor.

(b) Defina a função `filtro :: Extracto -> [String] -> [(Data,Movimento)]` que retorna informação relativa apenas aos movimentos cuja descrição esteja incluída na lista fornecida no segundo parâmetro.

(c) Defina a função `creDeb :: Extracto -> (Float,Float)`, que retorna o total de créditos e de débitos de um extracto no primeiro e segundo elementos de um par, respectivamente.

(d) Defina a função `saldo :: Extracto -> Float` que devolve o saldo final que resulta da execução de todos os movimentos no extracto sobre o saldo inicial.