Ficha 7

Programação Funcional

LEI 1º ano

1. 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 Eq

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

   (a) Defina a função `procura :: Nome -> TabDN -> Maybe Data`, que indica a data de uma dada pessoa, caso ela 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.

2. Considere as seguintes definições de tipos de dados para representar uma tabela de abreviaturas

```
type TabAbrev = [(Abreviatura,Palavra)]
type Abreviatura = String
type Palavra = String
```

   (a) Defina a função `daPal :: TabAbrev -> Abreviatura -> Maybe Palavra`, que dadas uma tabela de abreviaturas e uma abreviatura, devolve a palavra correspondente.

   (b) Analise a seguinte função que pretende transformar um texto, substituindo as abreviaturas que lá ocorrem pelas palavras correspondentes.

```
transforma :: TabAbrev -> String -> String
transforma t s = unwords (trata t (words s))
```

   Apresente uma definição adequada para a função `trata` e indique o seu tipo.

3. 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 valore são sempre números positivos).

```
data Movimento = Credito Float | Debito Float
data Extracto = Ext Float [(Data, String, Movimento)]
```

   (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. (Tente usar um `foldr` na sua implementação).

   (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. (Tente usar um `foldr` na sua implementação).