

# Ficha 7

## Programação Funcional

2015/16

1. 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).
2. 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.
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 valores são sempre números positivos).

```
data Movimento = Credito Float | Debito Float
  deriving Show
data Data = D Int Int Int      -- Dia Mes Ano
  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 de novo a função `creDeb :: Extracto -> (Float,Float)` usando um `foldr`.
- (e) 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.
- (f) Defina de novo a função `creDeb :: Extracto -> (Float,Float)` usando um `foldr`.