Programação Funcional

2013/14

Caderno de Exercícios

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:

   (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 `Data.Char` .

8. Usando as funções do módulo `Data.Char`

   (a) Defina a função `primMai`, e o seu tipo, que recebe uma string como argumento e testa se o seu primeiro caracter é uma letra maiúscula.

   (b) Defina a função `segMin`, e o seu tipo, que recebe uma string como argumento e testa se o seu segundo caracter é uma letra minúscula.

9. 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.

10. 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
    ```

2 Funções recursivas

11. Indique como é que o interpretador de haskell avalia as expressões das alíneas que se seguem, apresentando a cadeia de redução de cada uma dessas expressões (i.e., os vários passos intermédios até se chegar ao valor final).

    (a) Considere a definição da seguinte função

    ```
    funA :: [Float] -> [Float]
    funA [] = []
    funA (h:t) = if h>=0 then h : (funA t)
                         else (funA t)
    ```

    Diga, justificando, qual é o valor de `funA [3,-5,0,-3,2]`.

    (b) Considere a definição da seguinte função

    ```
    funB :: [Float] -> Float
    funB [] = 1
    funB (x:xs) = x * (funB xs)
    ```

    Diga, justificando, qual é o valor de `funB [3,5,2]`.

    (c) Considere a definição da seguinte função

    ```
    funC :: [Float] -> Float
    funC [] = 0
    funC (y:ys) = y^2 + (funC ys)
    ```

    Diga, justificando, qual é o valor de `funC [2,3,5]`.

    (d) Considere a definição da seguinte função

    ```
    funD :: [Int] -> [Int]
    funD [] = []
    funD (h:t) = if (mod h 2)==0 then h : (funD t)
                                 else (funD t)
    ```

    Diga, justificando, qual é o valor de `funD [8,5,12,7]`

    (e) Considere a seguinte definição

    ```
    p :: Int -> Bool
    p 0 = True
    p 1 = False
    p x | x > 1 = p (x-2)
    ```

    Diga, justificando, qual é o valor de `p 5`.

    (f) Considere a seguinte definição

    ```
    f l = g [] l
    g l [] = l
    g l (h:t) = g (h:l) t
    ```

    Diga, justificando, qual é o valor de `f "otrec"` ?

12. 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 do exercício 7.)

(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.

(h) Defina a função `somaNeg :: [Int] -> Int` que soma todos os números negativos da lista de entrada.

(i) Defina a função `soDigitos :: [Char] -> [Char]` que recebe uma lista de caracteres, e selecciona dessa lista os caracteres que são algarismos. Pode usar as funções no módulo `Data.Char`.

(j) Defina a função `minusculas :: [Char] -> Int` que recebe uma lista de caracteres, e conta quantos desses caracteres são letras minúsculas. Pode usar as funções no módulo `Data.Char`.

(k) `ordena :: [a] -> [a]` que ordena uma lista.

13. Considere que se definiu em Haskell o tipo `Jogo` de modo a definir resultados de jogos de futebol.

```
type Jogo = (String,Int,String,Int)    -- (Eq.casa,golos,Eq.visitante,golos)
```

(a) Escreva uma função `golosEquipa` (e o seu tipo), que dado um jogo e o nome de uma equipa, dá como resultado os golos que essa equipa marcou. Caso a equipa não tenha participado no jogo então a função devolve o valor `-1`.

(b) Escreva uma função `resJogo` (e o seu tipo), que dado um jogo devolve um character com valor `'1'` (indicando que ganhou a equipa da casa), `'x'` (indicando empate) e `'2'` (vitário da equipa visitante). Por exemplo, `resJogo ("Alemanha",2,"Portugal",3)` dá como resultado `'2'`.

(c) Escreva uma função `resJogo` (e o seu tipo), que dado um jogo devolve uma string que indica se ganhou a equipa da casa, a equipa visitante, ou se empataram. Por exemplo, `resJogo ("Alemanha",2,"Portugal",3)` dá como resultado a string `"ganhou equipa visitante"`.

(d) Escreva uma função, e o seu tipo, que dado uma lista de jogos indica quantos jogos terminaram em empate.

(e) Escreva a função `jogosComXGolos:: [Jogo] -> Int -> Int`, que dado uma lista de jogos e um número de golos, indica o número de jogos com esse número de golos.

(f) Escreva uma função, e o seu tipo, que dado uma lista de jogos indica quantos jogos foram ganhos por equipadas visitantes.

14. Considere as seguintes definições de tipo para representar círculos (guardando o centro e raio)

```
type Ponto = (Float, Float)      -- (Abcissa, Ordenada)
type Circulo = (Ponto, Float)    -- (Centro, Raio)
```

(a) Defina uma função `fora :: Ponto -> Circulo -> Bool` que testa se um ponto está fora de um círculo (i.e., se distância ao centro é maior do que o raio).

(b) Defina uma função `filtraFora :: Circulo -> [Ponto] -> Int` que, dado um círculo e uma lista de pontos, determina quantos pontos da lista estão fora do círculo dado (use, se precisar, a função anterior).

(c) Defina uma função `dentro :: Ponto -> Circulo -> Bool` que testa se um ponto está dentro de um círculo (i.e., se a distância ao centro é menod do que o raio).

(d) Defina uma função `filtraDentro :: Ponto -> [Circulo] -> Int` que, dado um ponto e uma lista de círculos, determina quantos círculos da lista contêm o ponto dado (use, se precisar, a função anterior).

15. Considere os seguintes tipos para representar pontos e rectângulos, respectivamente. Assuma que os rectângulos têm os lados paralelos aos eixos e são representados apenas por dois dos pontos mais afastados.

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

(a) Defina as seguintes funções:

- `quadrado :: Rectangulo -> Bool` que testa se um rectângulo é um quadrado.
- `contaQuadrados :: [Rectangulo] -> Int` que, dada uma lista com rectângulos, conta quantos deles são quadrados.

(b) Defina as seguintes funções:

- `roda :: Rectangulo -> Rectangulo` que roda um rectângulo 90º (centrado no primeiro ponto).
- `rodaTudo :: [Rectangulo] -> [Rectangulo]` que, dada uma lista com rectângulos, roda todos os rectângulos de acordo com a definição anterior.

(c) Defina as seguintes funções:

- `area :: Rectangulo -> Float` que determina a área de um rectângulo.
- `areaTotal :: [Rectangulo] -> Float` que, dada uma lista com rectângulos, determina a área total que eles ocupam.

(d) Defina as seguintes funções:

- `escala :: Float -> Rectangulo -> Rectangulo` que escala um rectângulo de acordo com um dado factor (mantendo o primeiro ponto).
- `escalaTudo :: Float -> [Rectangulo] -> [Rectangulo]` que, dada um factor e uma lista com rectângulos, escala todos os rectângulos de acordo com a definição anterior.

16. Defina as seguintes funções sobre números inteiros não negativos:

(a) `(><) :: Int -> Int -> Int` para multiplicar dois números inteiros (por somas sucessivas).

(b) `div, mod :: Int -> Int -> Int` que calculam a divisão e o resto da divisão inteiras por subtracções sucessivas.

(c) `power :: Int -> Int -> Int` que calcula a potência inteira de um número por multiplicações sucessivas.

(d) `uns :: Int -> Int` que calcula quantos bits 1 são usados para representar um número.

(e) `primo :: Int -> Bool` que testa se um número é primo.

17. Defina as seguintes funções sobre listas de pares:

(a) `primeiros :: [(a,b)] -> [a]` que calcula a lista das primeiras componentes.
Por exemplo, `primeiros [(10,21), (3, 55), (66,3)] = [10,3,66]`

(b) `nosPrimeiros :: a -> [(a,b)] -> Bool` que testa se um elemento aparece na lista como primeira componente de algum dos pares.

(c) `minFst :: (Ord a) => [(a,b)] -> a` que calcula a menor primeira componente.
Por exemplo, `minFst [(10,21), (3, 55), (66,3)] = 3`

(d) `sndMinFst :: (Ord a) => [(a,b)] -> b` que calcula a segunda componente associada à menor primeira componente.
Por exemplo, `sndMinFst [(10,21), (3, 55), (66,3)] = 55`

(e) `ordenaSnd :: [(a,b)] -> [(a,b)]` que ordena uma lista por ordem crescente da segunda componente.

3 Problemas (parte 1)

18. 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 de um aluno com nota mais alta.

19. 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 no exercício 9, 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).

20. Considere as seguintes definições.

```
type Ponto = (Float,Float)            -- (abcissa,ordenada)
type Rectangulo = (Ponto,Float,Float) -- (canto sup.esq., larg, alt)
type Triangulo = (Ponto,Ponto,Ponto)
type Poligonal = [Ponto]

distancia :: Ponto -> Ponto -> Float
distancia (a,b) (c,d) = sqrt (((c-a)^2) + ((b-d)^2))
```

(a) Defina uma função que calcule o comprimento de uma linha poligonal.

(b) Defina uma função que converta um elemento do tipo `Triangulo` na correspondente linha poligonal.

(c) Repita o alínea anterior para elementos do tipo `Rectangulo`.

(d) Defina uma função `fechada` que testa se uma dada linha poligonal é ou não fechada.

(e) Defina uma função `triangula` 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.

(f) Suponha que existe uma função `areaTriangulo` que calcula a área de um triângulo.

```
areaTriangulo (x,y,z)
   = let a = distancia x y
         b = distancia y z
         c = distancia z x
         s = (a+b+c) / 2 -- semi-perimetro
     in -- formula de Heron
        sqrt (s*(s-a)*(s-b)*(s-c))
```

Usando essa função, defina uma função que calcule a área delimitada por uma linha poligonal fechada e convexa.

(g) Defina uma função `mover` 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. Por exemplo, ao mover o triângulo `[(1,1),(10,10),(10,1),(1,1)]` para o ponto (1,2) devemos obter o triângulo `[(1,2),(10,11),(10,2),(1,2)]`.

(h) Defina uma função `zoom2` que, dada 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 por 2. Por exemplo, o rectângulo

```
[(1,1),(1,3),(4,3),(4,1),(1,1)]
```

deverá ser transformado em `[(1,1),(1,5),(7,5),(7,1),(1,1)]`

21. Considere que a informação sobre um stock de uma loja está armazenada numa lista de tuplos (com o nome do produto, o seu preço unitário, e a quantidade em stock desse produto) de acordo com as seguintes declarações de tipos:

```
type Stock = [(Produto,Preco,Quantidade)]
type Produto = String
type Preco = Float
type Quantidade = Float
```

Assuma que um produto não ocorre mais do que uma vez na lista de stock.

(a) Defina as seguintes funções de manipulação e consulta do stock:

i. `quantos::Stock->Int`, que indica quantos produtos existem em stock.

ii. `emStock::Produto->Stock->Quantidade`, que indica a quantidade de um dado produto existente em stock.

iii. `consulta::Produto->Stock->(Preco,Quantidade)`, que indica o preço e a quantidade de um dado produto existente em stock.

iv. `tabPrecos::Stock->[(Produto,Preco)]`, para construir uma tabela de preços.

v. `valorTotal::Stock->Float`, para calcular o valor total do stock.

vi. `inflacao::Float->Stock->Stock`, que aumenta uma dada percentagem a todos os preços.

vii. `omaisBarato::Stock->(Produto,Preco)`, que indica o produto mais barato e o seu preço.

viii. `maisCaros::Preco->Stock->[Produto]`, que constroi a lista dos produtos caros (i.e., acima de um dado preço).

(b) Considere agora que tem a seguinte declaração de tipo para modelar uma lista de compras:

```
type ListaCompras = [(Produto,Quantidade)]
```

Defina as funções que se seguem:

i. `verifLista::ListaCompras->Stock->Bool`, que verifica se todos os pedidos podem ser satisfeitos.

ii. `falhas::ListaCompras->Stock->ListaCompras`, que constroi a lista dos pedidos não satisfeitos.

iii. `custoTotal::ListaCompras->Stock->Float`, que calcula o custo total da lista de compras.

iv. `partePreco::Preco->ListaCompras->Stock->(ListaCompras,ListaCompras)`, que parte a lista de compras em duas: uma lista com os items inferiores a um dado preço, e a outra com os items superiores ou iguais a esse preço.

22. Um multi-conjunto é um conjunto que admite elementos repetidos. É diferente de uma lista porque a ordem dos elementos não é relevante. Uma forma de implementar multi-conjuntos em Haskell é através de uma lista de pares, onde cada par regista um elemento e o respectivo número de ocorrências:

```
type MSet a = [(a,Int)]
```

Uma lista que representa um multi-conjunto não deve ter mais do que um par a contabilizar o número de ocorrências de um elemento, e o número de ocorrências deve ser sempre estritamente positivo. O multi-conjunto de caracteres `{'b','a','c','a','b','a'}` poderia, por exemplo, ser representado pela lista `[('b',2),('a',3),('c',1)]`.

(a) Defina a função `elem :: Eq a => a -> MSet a -> Bool` que testa se um determinado elemento pertence a um multi-conjunto. Por exemplo,

```
> elem 'b' [('b',2),('a',3),('c',1)]
True
> elem 'd' [('b',2),('a',3),('c',1)]
False
```

(b) Defina a função `converte :: Eq a => [a] -> MSet a` que converte uma lista para um multi-conjunto. Por exemplo,

```
> converte "bacaba"
[('b',2),('a',3),('c',1)]
```

(c) Defina a função `size : MSet a -> Int` que calcula o tamanho de um multi-conjunto.

```
> size [('b',2),('a',3),('c',1)]
6
```

(d) Defina a função `union :: Eq a => MSet a -> MSet a -> MSet a` que calcula a união de dois multi-conjuntos. Por exemplo,

```
> union [('a',3),('b',2),('c',1)] [('d',5),('b',1)]
[('a',3),('b',3),('c',1),('d',5)]
```

(e) Defina a função `elimina :: Eq a => a -> MSet a -> MSet a` que elimina um elemento de um multi-conjunto. Por exemplo,

```
> elimina 'a' [('b',2),('a',3),('c',1)]
[('b',2),('a',2),('c',1)]
> elimina 'c' [('b',2),('a',3),('c',1)]
[('b',2),('a',3)]
```

(f) Defina a função `ordena :: MSet a -> MSet a` que ordena um multi-conjunto pelo número crescente de ocorrências. Por exemplo,

```
> ordena [('b',2),('a',3),('c',1)]
[('c',1),('b',2),('a',3)]
```

(g) Defina a função `insere :: Eq a => a -> MSet a -> MSet a` que insere um elemento num multi-conjunto. Por exemplo,

```
> insere 'a' [('b',2),('a',3),('c',1)]
[('b',2),('a',4),('c',1)]
> insere 'd' [('b',2),('a',3),('c',1)]
[('b',2),('a',3),('c',1),('d',1)]
```

(h) Defina a função `moda :: MSet a -> [a]` que devolve a lista dos elementos com maior número de ocorrências. Por exemplo,

```
> moda [('b',2),('a',3),('c',1),('d',3)]
['a','d']
```

23. Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

```
type Radar = [(Hora,Matricula,VelAutor,VelCond)
type Hora = (Int,Int)      -- (horas, minutos)
type Matricula = String    -- matricula do carro em infraccao
type VelAutor = Int        -- velocidade autorizada
type VelCond  = Float      -- velocidade do condutor
```

(a) Escreva a função, e seu tipo, que dado a matricula de um carro, calcula o excesso de velocidade desse carro nesse dia. Note que um carro pode ter mais do que uma infração.

(b) Escreva uma função que recebe a primeira componente do par `Hora` e devolve quantas infrações se realizaram nesse período de uma hora.

(c) Considere que o radar deve registar as infrações por ordem crescente da hora. Defina uma função que verifica se o radar está a funcionar correctamente. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`horaMaior :: Hora -> Hora -> Bool`).

(d) Escreva uma função, e o seu tipo, que verifica se o radar registou duas infrações à mesma hora.

(e) Escreva uma função que calcula a maior infração registada (*i.e.*, maior diferença entre velocidade do condutor e autorizada).

(f) Escreva a função que calcula o menor período de tempo (em minutos) sem infrações. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`hora2mins`).

(g) Escreva uma função, e o seu tipo, que verifica se houve algum carro apanhado em excesso de velocidade mais do que uma vez.

(h) Escreva a função que dado a matricula de um carro, devolve um lista com as infrações desse carro. Esta lista contém pares com a hora e a **velocidade em excesso do carro**. Note que um carro pode ser apanhado em excesso mais do que uma vez no mesmo dia.

(i) Considere que o radar deve registar as infrações por ordem crescente da hora. Defina uma função que verifica se o radar está a funcionar correctamente. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`horaMaior :: Hora -> Hora -> Bool`).

(j) Escreva uma função, e o seu tipo, que verifica se o radar está a funcionar correctamente (isto é, a velocidade do condutor é sempre maior que a velocidade autorizada).

(k) Escreva uma função que calcula o total de excesso de velocidade nesse dia.

(l) Escreva a função que calcula o maior período de tempo sem infrações durante o dia. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`hora2mins`).

24. Considere as seguintes definições de tipos para representar os alunos inscritos na UM.

```
type Inscritos = [(Num,Nome,Curso,Ano)]
type Num = Integer
type Nome = String
type Curso = String
type Ano = Integer
```

   (a) Defina a função `aluCA :: (Curso, Ano) -> Inscritos -> Int`, que calcula o número de alunos inscritos num determinado ano de um dado curso.

   (b) Defina a função `quantos :: Curso -> [Num] -> Inscritos -> Int` que, dado um curso c, uma lista de números l e uma tabela de inscritos t, calcula quantos números da lista l correspondem a alunos inscritos no curso c.

   (c) Defina a função `doAno :: Ano -> Inscritos -> [(Num, Nome, Curso)]`, que seleciona todos os alunos que frequentam um determinado ano.

25. Considere as seguintes definições de tipos para representar uma *playlist* de músicas.

```
type PlayList = [(Titulo,Interprete,Duracao)]
type Titulo = String
type Interprete = String
type Dutacao = Int       -- duração da música em segundos
```

   (a) Defina a função `total :: PlayList-> Int`, que calcula ao tempo total da *playlist*.

   (b) Defina a função `temMusicas :: [Interprete] -> PlayList -> Bool`, tal que testa se todos os intérpretes que aparecem a lista têm alguma música na *playlist*.

   (c) Defina a função `maior :: PlayList-> (Titulo, Duracao)`, que indica o titulo e a duração de uma das músicas de maior duração da *playlist*.

26. Considere as seguintes definções de tipos para representar uma tabela de abreviaturas que associa a cada abreviatura a palavra que ela representa.

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

   (a) Defina a função `existe :: Abreviatura -> TabAbrev -> Bool`, que verifica se uma dada abreviatura existe na tabela.

   (b) Defina a função `substitui :: [String] -> TabAbrev -> [String]`, que recebe um texto (dado como uma lista de stings) e uma tabela de abreviaturas, substitui todas as abreviaturas que apareçam no texto pelas respectivas palavras associadas.

   (c) Defina a função `estaOrdenada :: TabAbrev -> Bool` que testa se a tabela de abreviaturas está ordenada por ordem crescente de abreviatura. (Nota: pode usar o operador $<$ para comparar directamente duas strings.)

27. Considere as seguintes definições de tipos para representar uma tabela de registo de temperaturas.

```
type TabTemp = [(Data,Temp,Temp)]  -- (data, temp. mínima, temp. máxima)
type Data = (Int,Int,Int)          -- (ano, mês, dia)
type Temp = Float
```

(a) Defina a função `médias :: TabTemp -> [(Data,Temp)]` que constroi a lista com as temperaturas médias de cada dia.

(b) Defina a função `decrescente :: TabTemp -> Bool` que testa se a tabela está ordenada por ordem decrescente de data. (Nota: pode usar o operador $>$ para comparar directamente duas datas.)

(c) Defina a função `conta :: [Data] -> TabTemp -> Int` que, dada uma lista de datas e a tabela de registo de temperaturas, conta quantas das datas da lista têm registo de na tabela.

28. Considere as seguintes definições de tipo para representar polinómios

```
type Monomio = (Float,Int)         -- (Coeficiente, Expoente)
type Polinomio = [Monomio]
```

Assuma que os polinómios têm no máximo um monómio para cada grau e que não são armazenados monómios com coeficiente nulo. Por exemplo, o polinómio $5\,x^3 + x - 5$ pode ser representado pelas listas `[(5,3),(1,1),(-5,0))]` ou `[(-5,0),(5,3),(1,1))]`.

Defina as seguintes funções:

(a) `coef :: Polinomio -> Int -> Float` que calcula o coeficiente de um dado grau (0 se não existir).

(b) `poliOk :: Polinomio -> Bool` que testa se um polinómio está bem construído (i.e., se não aparecem monómios com graus repetidos nem coeficientes nulos).

(c) `addM :: Polinomio -> Monomio -> Polinomio` que adiciona um polinómio a um monómio. Não se esqueça de garantir que o polinómio resultante não tem monómios de grau repetido nem coeficientes nulos.

(d) `addP :: Polinomio -> Polinomio -> Polinomio` que adiciona dois polinómios.

4 Funções recursivas que devolvem "tuplos"

29. A função `divMod :: Int -> Int -> (Int,Int)`, já predefinida no Prelude, poderia ser definida pela seguinte equação:

```
divMod x y = (div x y, mod x y)
```

Apresente uma definição alternativa desta função sem usar `div` e `mod` como funções auxiliares.

30. Defina uma função `nzp :: [Int] -> (Int,Int,Int)` que, dada uma lista de inteiros, conta o número de valores nagativos, o número de zeros e o número de valores positivos, devolvendo um triplo com essa informação. Certifique-se que a função que definiu percorre a lista apenas uma vez.

31. Defina a função `semSep :: String -> (String,Int)`, que dada uma string, lhe retira os separadores e conta o número de caracteres da string resultante. Implemente a função de modo a fazer uma única travessia da string. (Relembre que a função `isSpace::Char->Bool` está já definida no módulo `Data.Char`).

32. Defina a função `digitAlpha :: String -> (String,String)`, que dada uma string, devolve um par de strings: uma apenas com as letras presentes nessa string, e a outra apenas com os números presentes na string. Implemente a função de modo a fazer uma única travessia da string. (Relembre que as funções `isDigit, isAlpha::Char->Bool` estão já definidas no módulo `Data.Char`).

5 Listas por compreensão

33. Para cada uma das expressões seguintes, exprima por enumeração a lista correspondente. Tente ainda, para cada caso, descobrir uma outra forma de obter o mesmo resultado.

    (a) `[x | x <- [1..20], mod x 2 == 0, mod x 3 == 0]`

    (b) `[x | x <- [y | y <- [1..20], mod y 2 == 0], mod x 3 == 0]`

    (c) `[(x,y) | x <- [0..20], y <- [0..20], x+y == 30]`

    (d) `[sum [y | y <- [1..x], odd y] | x <- [1..10]]`

34. Defina cada uma das listas seguintes por compreensão.

    (a) `[1,2,4,8,16,32,64,128,256,512,1024]`

    (b) `[(1,5),(2,4),(3,3),(4,2),(5,1)]`

    (c) `[[1],[1,2],[1,2,3],[1,2,3,4],[1,2,3,4,5]]`

    (d) `[[1],[1,1],[1,1,1],[1,1,1,1],[1,1,1,1,1]]`

    (e) `[1,2,6,24,120,720]`

6 Funções de ordem superior

35. Apresente definições das seguintes funções de ordem superior, já predefinidas no Perlude:

    (a) `zipWith :: (a->b->c) -> [a] -> [b] -> [c]` que combina os elementos de duas listas usando uma função específica; por exemplo `zipWith (+) [1,2,3,4,5] [10,20,30,40] = [11,22,33,44]`.

    (b) `takeWhile :: (a->Bool) -> [a] -> [a]` que determina os primeiros elementos da lista que satisfazem um dado predicado; por exemplo `takeWhile odd [1,3,4,5,6,6] = [1,3]`.

    (c) `dropWhile :: (a->Bool) -> [a] -> [a]` que elimina os primeiros elementos da lista que satisfazem um dado predicado; por exemplo `dropWhile odd [1,3,4,5,6,6] = [4,5,6,6]`.

    (d) `span :: (a-> Bool) -> [a] -> ([a],[a])`, que calcula simultaneamente os dois resultados anteriores. Note que apesar de poder ser definida à custa das outras duas, usando a definição

    ```
    span p l = (takeWhile p l, dropWhile p l)
    ```

    nessa definição há trabalho redundante que pode ser evitado. Apresente uma definição alternativa onde não haja duplicação de trabalho.

36. Defina a função `agrupa :: String -> [(Char,Int)]` que dada uma string, junta num par `(x,n)` as `n` ocorrências consecutivas de um caracter `x`. Por exemplo, `agrupa ‘‘aaakkkkkwaa’’` deve dar como resultado a lista `[('a',3), ('k',4), ('w',1), ('a',2)]`.

37. Considere a seguinte definição usando listas por compreensão:

    ```
    prod :: [a] -> [b] -> [(a,b)]
    prod l1 l2 = [(a,b) | a <- l1, b <- l2]
    ```

    Por exemplo,

    ```
    prod [1,2] ['a','b','c'] = [(1,'a'),(1,'b'),(1,'c'),(2,'a'),(2,'b'),(2,'c')]
    ```

De forma a definir esta função sem usar o mecanismo de definição de listas por compreensão, vamos defini-la à custa de outras funções auxiliares:

(a) A função `criaLinhas :: [a] -> [b] -> [[(a,b)]]` aplica a função anterior a cada elemento da primeira lista.

Por exemplo,

```
criaLinhas [1,2] ['a','b','c'] =
[[(1,'a'),(1,'b'),(1,'c')],[(2,'a'),(2,'b'),(2,'c')]]
```

i. Apresente uma definição (explicitamente) recursiva da função `criaLinhas`.

ii. Complete a seguinte definição da função `criaLinhas` (comece por determinar o tipo da função `f`)

```
crialinhas l1 l2 = map (f l2) l1
where f l x = ...
```

iii. Finalmente, podemos definir a função pretendida, concatenando todas as linhas produzidas pela função anterior.

```
prod l1 l2 = concat (criaLinhas l1 l2)
```

Apresente uma definição da função `concat :: [[a]] -> [a]` que concatena uma lista de listas numa lista só.

(b) A função `criaPares :: a -> [b] -> [(a,b)]` recebe um elemento do tipo `a` e uma lista e cria uma lista de pares cuja primeira componente é sempre a mesma.

Por exemplo, `criaPares 1 ['a','b','c'] = [(1,'a'),(1,'b'),(1,'c')]`

i. Apresente uma definição (explicitamente) recursiva da função `criaPares`.

ii. Complete a seguinte definição da função `criaPares` (comece por determinar o tipo da função `acrescenta`)

```
criaPares a bs = map (acrescenta a) bs
    where acrescenta a x = ...
```

iii. Complete a seguinte definição de forma a que a função `prod'` seja equivalente a `prod`.

```
prod' l1 l2 = concat (map ... l1)
```

38. Considere a função seguinte

```
indicativo  :: String -> [String] -> [String]
indicativo ind telefs = filter (concorda ind) telefs
  where concorda :: String -> String -> Bool
        concorda [] _ = True
        concorda (x:xs) (y:ys) = (x==y) && (concorda xs ys)
        concorda (x:xs) [] = False
```

que recebe uma lista de algarismos com um indicativo, uma lista de listas de algarismos representando números de telefone, e seleciona os números que começam com o indicativo dado. Por exemplo:

```
indicativo "253" ["253116787","213448023","253119905"]
```

devolve `["253116787","253119905"]`.

Redefina esta função com recursividade explícita, isto é, evitando a utilização de `filter`.

39. Defina uma função `toDigits :: Int -> [Int]` que, dado um número (na base 10), calcula a lista dos seus dígitos (por ordem inversa). Por exemplo, `toDigits 1234` deve corresponder a `[4,3,2,1]`. Note que

$$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

40. Pretende-se agora que defina a função inversa da anterior `fromDigits :: [Int] -> Int`. Por exemplo, `fromDigits [4,3,2,1]` deve corresponder a `1234`.

    (a) Defina a função com auxílio da função `zipWith`.

    (b) Defina a função com recursividade explícita. Note que

$$\begin{aligned} \texttt{fromDigits [4,3,2,1]} &= 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 \\ &= 4 + 10 \times (3 + 10 \times (2 + 10 \times (1 + 10 \times 0))) \end{aligned}$$

    (c) Defina agora a função usando um `foldr`.

41. Usando as funções anteriores e as funções do módulo `Data.Char`, `intToDigit :: Int -> Char` e `digitToInt :: Char -> Int`:

    (a) Defina a função `intStr :: Int -> String` que converte um inteiro numa string. Por exemplo, `intStr 1234` deve corresponder à string `"1234"`.

    (b) Defina a função `strInt :: String -> Int` que converte a representação de um inteiro (em base 10) nesse inteiro. Por exemplo, `strInt "12345"` deve corresponder ao número `12345`.

42. Defina a função `subLists :: [a] -> [[a]]` que calcula todas as sublistas de uma lista; por exemplo, `subLists [1,2,3] = [[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]`.

43. Assuma que a informação sobre os resultados dos jogos de uma jornada de um campeonato de futebol está guardada na seguinte estrutura de dados:

```
type Jornada = [Jogo]
type Jogo = ((Equipa,Golos),(Equipa,Golos))     -- (eq. casa, eq. visitante)
type Equipa = String
type Golos = Int
```

    (a) Defina a função `totalGolos :: Jornada -> Int` que calcula o total de golos da jornada.

    (b) Defina a função `numGolos :: Int -> Jornada -> [Jogo]` de forma a que `(numGolos x j)` represente a lista de jogos com mais de `x` golos marcados. De preferência, use funções de ordem superior.

    (c) Considere a seguinte função:

```
venceCasa :: Jornada -> [Equipa]
venceCasa j = map casa (filter vc j)
```

    Defina as funções `vc` e `casa` de forma a que a função `venceCasa` calcule a lista das equipas que venceram em casa numa dada jornada.

    (d) Defina a função `pontos :: Jornada -> [(Equipa,Int)]` que calcula os pontos que cada equipa obteve na jornada (venceu - 3 pontos; perdeu - 0 pontos; empatou - 1 ponto)

(e) Defina a função `empates :: Jornada -> [Jogo]` que seleciona os jogos da jornada em que ocorreram empates. De preferência, use funções de ordem superior.

(f) Considere a seguinte função:

```
golosMarcados :: Jornada -> Int
golosMarcados j = sum (map soma j)
```

Defina a função `soma` de forma a que a função `golosMarcados` calcule o número total de golos marcados numa jornada.

44. Apresente uma definição alternativa das seguintes funções, usando recursividade explícita em vez de funções de ordem superior.

(a)
```
func1 :: [[a]] -> [Int]
func1 l = map length (filter null l)
```

(b)
```
func2 :: [a] -> [a] -> Bool
func2 l m = and (zipWith (==) l m)
```

(c)
```
func3 :: Ord a => a -> [a] -> Int
func3 x l = length (filter (>= x) l)
```

45. Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

```
type Radar = [(Hora,Matricula,VelAutor,VelCond)]
type Hora = (Int,Int)
type Matricula = String    -- matricula do carro em infraccao
type VelAutor = Int        -- velocidade autorizada
type VelCond  = Float      -- velocidade do condutor
```

(a) Geralmente a GNR considera que a velocidade registada do condutor pode ter um erro de 10% (em excesso). Escreva uma função de ordem superior que dado as infrações registadas pelo radar num dia, aplica esta tolerância à velocidade registada.

(b) Escreva uma função de ordem superior que dado a matricula de um carro e as insfrações registadas num dia, devolve uma lista com as infrações desse carro. Note que um carro pode ser apanhado em excesso mais do que uma vez no mesmo dia.

(c) Escreva uma função que dado os registos de infrações de um dia, devolve a lista com a matricula do carro e o excesso de velocidade a que se deslocava. O excesso de velocidade para um registo é a diferença entre velocidade real (após aplicar a tolerência) e velocidade autorizada.

(d) Escreva um função calcula o valor total apurado em multas por excesso de velocidade num dia. Assuma que o valor das multas é: 60Eur para excessos não superiores 20Km/h; 120Eur para excessos entre 21 e 40 Km/h; 300Eur para excessos entre 41 e 60 Km/h; e 500Eur para excessos superiores a 60Km/h.

46. Considere que a GNR, para registar os teste de alcoolemia que realiza nas suas operações stop, utiliza a seguinte estrutura de dados:

```
type RegAlcool = [(Nome,Sexo,Idade,NA)]
type Nome  = String
type Sexo  = Char      -- 'M': Masculino 'F': Feminino
type Idade = Int
type NA    = Float     -- Nivel de Alcool
```

(a) Escreva uma função de ordem superior que dados os testes realizados, devolve informação dos testes realizados a pessoas menores de 21 anos.

(b) Considerando que a multa a pagar para quem conduzir com uma taxa no sangue superior a `0.5` se calcula de acordo com a regra `Nível de Álcool * 100 EUR`, escreva uma função que dados os testes realizados, produz uma lista onde se indica: o nome e o valor da multa a pagar (`0 EUR` caso esteja legal).

(c) Defina uma função que cálcula a média de idades das pessoas que fizeram o teste de alcoolemia.

(d) Escreva uma função de ordem superior que dados os testes realizados, devolve os testes realizados a mulheres apenas.

(e) Considerando que a lei autoriza a condução com uma taxa de alcoól no sangue de `0.5`, escreva uma função que dados os testes realizados, produz uma lista onde se indica: o nome da pessoa (que fez o teste) e uma string a indicar se a condução é ``` ``legal'' ``` ou ``` ``ilegal'' ```.

47. Uma forma de representar polinómios de uma variável é usar listas de monómios representados por pares *(coeficiente, expoente)*

```
type Polinomio = [Monomio]
type Monomio = (Float,Int)
```

Por exemplo, `[(2,3), (3,4), (5,3), (4,5)]` representa o polinómio $2\,x^3 + 3\,x^4 + 5\,x^3 + 4\,x^5$.

(a) Defina uma função `conta :: Int -> Polinomio -> Int` de forma a que `(conta n p)` indica quantos monómios de grau `n` existem em `p`.

(b) Defina uma função `selgrau :: Int -> Polinomio -> Polinomio` de forma a que `(selgrau n p)` que selecione do polinómio `p` os monómios de grau superior a `n`. De preferência, use funções de ordem superior.

(c) Complete a definição da função `deriv` de forma a que esta calcule a derivada de um polinómio.

```
deriv :: Polinomio -> Polinomio
deriv p = map ........ p
```

(d) Defina uma função `calcula :: Float -> Polinomio -> Float`, que calcula o valor do polinómio num dado valor de x.

(e) Defina a função `simp :: Polinomio -> Polinomio` que retira de um polinómio os monómios de coeficiente zero. De preferência, use funções de ordem superior.

(f) Complete a definição da função `mult` de forma a que esta calcule o resultado da multiplicação de um monómio por um polinómio.

```
mult :: Monomio -> Polinomio -> Polinomio
mult (c,e) p = map ........ p
```

7 Problemas numéricos

48. Considere a sequinte definição para representar matrizes:

```
type Mat a = [[a]]
```

Defina as seguintes funções sobre matrizes (use, sempre que achar apropriado, funções de ordem superior).

(a) `dimOK :: Mat a -> Bool` que testa se uma matriz está bem construída (i.e., se todas as linhas têm a mesma dimensão).

(b) `dimMat :: Mat a -> (Int,Int)` que calcula a dimensão de uma matriz.

(c) `addMat :: Mat a -> Mat a -> Mat a` que adiciona duas matrizes.

(d) `transpose :: Mat a -> Mat a` que calcula a transposta de uma matriz.

(e) `multMat :: Mat a -> Mat a -> Mat a` que calcula o produto de duas matrizes.

49. O *Crivo de Eratóstenes* é um método simples e prático para encontrar números primos até um certo valor limite. Pedemos descreve-lo assim:

- começamos com a lista de inteiros entre 2 (o primeiro primo) e o valor limite;
- destacamos o primeiro elemento da lista (que irá ficar na lista de saida), retiramos da cauda da lista todos os multiplos dele, e continuamos a processar a caula da lista (já filtrada) pelo mesmo método.

Defina uma função que implemente este algoritmo

50. Um multiconjunto é um conjunto em que a multiplicidade é relevante.

São por isso diferentes os multiconjuntos: $\{1,2,3,4,1,2,1\}$ e $\{1,2,3,4\}$. Considere que se usa o seguinte tipo para representar multi-conjuntos:

```
type MSet a = [(a,Int)]
```

Neste tipo, o multiconjunto `{'a','c','a','b','c','a'}` é representado por `[('a',3), ('b',1), ('c', 2)]`. Considere ainda que estas listas estão ordenadas (pela primeira componente) e que não há pares cuja primeira componente coincida.

(a) Defina a função `add :: Ord a => a -> (MSet a) -> MSet a` que acrescenta um elemento a um multiconjunto.

(b) Defina a função `toMSet :: (Ord a) => [a] -> MSet a`que constroi um multiconjunto com os elementos de uma lista.

(c) Defina uma função `moda :: MSet a -> a` que determina qual o elemento mais frequente de uma multiconjunto (não vazio).

(d) Defina as função `mIntersect, mUnion :: (Ord a) => (MSet a) -> (MSet a) -> (MSet a)` de intersecção e união de multiconjuntos.

51. O *Teorema Fundamental da Aritmética* (enunciado pela primeira vez por Euclides) diz que qualquer número inteiro (maior do que 1) pode ser decomposto num produto de números primos. Esta decomposição é além disso única a menos de uma permutação. Por exemplo,

$$212121 = 3 \times 3 \times 7 \times 7 \times 13 \times 37$$

$$222222 = 2 \times 3 \times 7 \times 11 \times 13 \times 37$$

(a) Defina uma função `factoriza :: Integer -> [Integer]` que, dado um número (maior do que 1) calcula a lista dos seus factores primos (por exemplo, `factoriza 212121` deve calcular a lista `[3,3,7,7,13,37]`).

(b) Dadas as factorizações de dois números é fácil calcular (as factorizações de) o máximo divisor comum (`mdc`) e o mínimo múltiplo comum (`mmc`).

- o máximo divisor comum obtém-se com *os factores comuns elevados à menor potência*. Assim,

$$\begin{aligned} \texttt{mdc}\ 212121\ 222222 &= \texttt{mdc}\ (3^2 \times 7^2 \times 13^1 \times 37^1)(2^1 \times 3^1 \times 7^1 \times 11^1 \times 13^1 \times 37^1) \\ &= 3^1 \times 7^1 \times 13^1 \times 37^1 \\ &= 10101 \end{aligned}$$

- o mínimo múltiplo comum obtém-se com *os factores comuns e não comuns elevados à maior potência*. Assim,

$$\begin{aligned} \texttt{mmc}\ 212121\ 222222 &= \texttt{mmc}\ (3^2 \times 7^2 \times 13^1 \times 37^1)(2^1 \times 3^1 \times 7^1 \times 11^1 \times 13^1 \times 37^1) \\ &= 2^1 \times 3^2 \times 7^2 \times 11^1 \times 13^1 \times 37^1 \\ &= 4666662 \end{aligned}$$

Defina as funções `mdcF, mmcF :: Integer -> Integer -> Integer` que calculam o máximo divisor comum e mínimo múltiplo comum usando as factorizações dos números em causa.

(c) Uma outra forma (muito mais eficaz) de calcular o máximo divisor comum entre dois números baseia-se na seguinte propriedade (também atribuída a Euclides):

$$\texttt{mdc}\ x\ y = \texttt{mdc}\ (x+y)\ y = \texttt{mdc}\ x\ (y+x)$$

Apresente uma definição da função `mdc` usando esta propriedade. Note ainda que a função `mmc` pode ser definida usando `mdc`:

```
mmc :: Integer -> Integer -> Integer
mmc x y = (x * y) `div` (mdc x y)
```

52. Uma representação possível de polimómios é pela sequência dos coeficientes - vamos ter de armazenar também os coeficientes nulos pois será a posição do coeficiente na lista que dará o grau do monómio. Teremos então

```
type Polinomio = [Coeficiente]
type Coeficiente = Float
```

A representação do polinómio $2\,x^5 - 5\,x^3$ será então

```
[0,0,0,-5,0,2]
```

que corresponde ao polinómio $0\,x^0 + 0\,x^1 + 0\,x^2 - 5\,x^3 + 0\,x^4 + 2\,x^5$.

(a) Defina a operação que calcula o valor do polinómio para um dado x.

(b) Defina a operação que calcula a derivada de um polinómio.

(c) Defina a operação de adição de polinómios.

(d) Defina a operação de multiplicação de polinómios.

8 Data types

53. 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 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.

54. 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.

55. 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
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).

56. Considere o seguinte tipo para representar árvores binárias.

```
data BTree a = Empty
             | Node a (BTree a) (BTree a)
            deriving Show
```

Defina as seguintes funções:

(a) `altura :: (BTree a) -> Int` que calcula a altura da árvore.

(b) `contaNodos :: (BTree a) -> Int` que calcula o número de nodos da árvore.

(c) `folhas :: (BTree a) -> Int`, que calcula o número de folhas (i.e., nodos sem descendentes) da árvore.

(d) `prune :: Int -> (Btree a) -> BTree a`, que remove de uma árvore todos os elementos a partir de uma determinada profundidade.

(e) `path :: [Bool] -> (BTree a) -> [a]`, que dado um caminho (`False` corresponde a *esquerda* e `True` a *direita*) e uma árvore, dá a lista com a informação dos nodos por onde esse caminho passa.

(f) `mirror :: (BTree a) -> BTree a,` que dá a árvore simétrica.

(g) `zipWithBT :: (a -> b -> c) -> (BTree a) -> (BTree b) -> BTree c` que generaliza a função `zipWith` para árvores binárias.

(h) `unzipBT :: (BTree (a,b,c)) -> (BTree a,BTree b,BTree c)`, que generaliza a função `unzip` (neste caso de triplos) para árvores binárias.

57. Considere o seguinte tipo para representar **árvores binárias de procura**.

```
data BTree a = Empty
             | Node a (BTree a) (BTree a)
            deriving Show
```

Defina as seguintes funções:

(a) i. Defina uma função `minimo :: (Ord a) => BTree a -> a` que calcula o menor elemento de uma árvore binária de procura **não vazia**.

ii. Defina uma função `semMinimo :: (Ord a) => BTree a -> BTree a` que remove o menor elemento de uma árvore binária de procura **não vazia**.

iii. Defina uma função `minSmin :: (Ord a) => BTree a -> (a,BTree a)` que calcula, com uma única consulta da árvore o resultado das duas funções anteriores.

(b) i. Defina uma função `maximo :: (Ord a) => BTree a -> a` que calcula o maior elemento de uma árvore binária de procura **não vazia**.

ii. Defina uma função `semMaximo :: (Ord a) => BTree a -> BTree a` que remove o maior elemento de uma árvore binária de procura **não vazia**.

iii. Defina uma função `maxSmax :: (Ord a) => BTree a -> (a,BTree a)` que calcula, com uma única consulta da árvore o resultado das duas funções anteriores.

58. Considere agora que guardamos a informação sobre uma turma de alunos na seguinte estructura de dados:

```
type Aluno = (Numero,Nome,Regime,Classificacao)
type Numero = Int
type Nome = String
data Regime = ORD | TE | MEL  deriving Show
data Classificacao = Aprov Int
                   | Rep
                   | Faltou
   deriving Show
type Turma = BTree Aluno    -- árvore binária de procura (ordenada por número)
```

Defina as seguintes funções:

(a) `inscNum :: Numero -> Turma -> Bool`, que verifica se um aluno, com um dado número, está inscrito.

(b) `inscNome :: Nome -> Turma -> Bool`, que verifica se um aluno, com um dado nome, está inscrito.

(c) `trabEst :: Turma -> [(Numero,Nome)]`, que lista o número e nome dos alunos trablhadores-estudantes (ordenados por número).

(d) `nota :: Numero -> Turma -> Maybe Classificacao`, que calcula a classificação de um aluno (se o aluno não estiver inscrito a função deve retornar `Nothing`).

(e) `percFaltas :: Turma -> Float`, que calcula a percentagem de alunos que faltaram à avaliação.

(f) `mediaAprov :: Turma -> Float`, que calcula a média das notas dos alunos que passaram.

(g) `aprovAv :: Turma -> Float`, que calcula o rácio de alunos aprovados por avaliados. Implemente esta função fazendo apenas uma travessia da árvore.

59. Considere que para representar ainformação dos concorrentes do Dakar se utilizou o seguinte tipo de dados:

```
type Dakar = [Piloto]

data Piloto = Carro  Numero Nome Categoria
            | Mota   Numero Nome Categoria
            | Camiao Numero Nome
type Numero = Int
type Nome   = String
data Categoria = Competicao | Maratona
```

(a) Assuma que a lista está ordenada pelo nome do piloto, escreva a função `inserePil :: Piloto -> Dakar -> Dakar` que insere ordenadamente um novo piloto na lista.

(b) Escreva uma função que verifica se a lista está ou não ordenada.

(c) Considere que para melhorar a procura de um piloto se utilizou uma árvore binária de procura, **ordenada pelo número**.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
type Dakar = BTree Piloto
```

i. Escreva uma função `maior :: Dakar -> Piloto` que calcula o piloto **com número maior**.

ii. Escreva uma função `menor :: Dakar -> Piloto` que calcula o piloto **com número menor**.

iii. Escreva uma função `listaMotas :: Dakar -> [(Numero, Nome)]` que lista ordenadamente (por número) o numero e o nome de todos os pilotos das motas.

60. 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

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).

61. Resolveu-se criar uma base de dados de vídeos e, para isso, classificaram-se os vídeos em: filmes, episódios de séries, shows, e outros. Para esse fim, definiram-se os seguintes tipos de dados.

```
type BD = [Video]
data Video = Filme String Int       -- título, ano
           | Serie String Int Int   -- título, temporada, episódio
           | Show  String Int       -- título, ano
           | Outro String
```

(a) Defina a função `espectaculos :: BD -> [(String,Int)]` que indica o título e ano de todos os espetáculos da base de dados.

(b) Defina a função `filmesAno :: Int -> BD -> [String]` que indica os títulos dos filmes de um dado ano que existem na base de dados.

(c) Defina a função `outros :: BD -> BD` que seleciona da base de dados todos os vídeos que não são filmes, séries ou shows.

(d) Defina a função `totalEp :: String -> BD -> Int` que indica quantos episódios de uma dada série existem na base de dados.

62. Considere que para organizar os seus livros foi definido o seguinte tipo de dados:

```
type Biblio = [Livro]

data Livro = Romance Titulo Autor Ano Lido
           | Ficcao  Titulo Autor Ano Lido
type Titulo = String
type Autor  = String
type Ano    = Int
data Lido   = Sim
            | Nao
```

(a) Escreva uma função que verifica se existem, ou não, livros repetidos.

(b) Defina a função `lido :: Biblio -> Titulo -> Biblio` que marca um dado livro como lido na biblioteca.

(c) Escreva uma função que calcula o número de livros lidos.

(d) Considere que a lista de livros está ordenada por ordem alfabética consoante o seu autor. Escreva uma função `compra :: Titulo -> Autor -> Ano -> Biblio -> Biblio` que insere ordenadamente o livro comprado (e não lido) na biblioteca.

(e) Considere que para melhorar a procura de um livro se utilizou uma árvore binária de procura, ordenada pelo título.

```
data BTree a = Vazia | Nodo a (BTree a) (BTree a)
type Biblio = BTree Livro
```

i. Escreva uma função `info :: Biblio -> Titulo -> Maybe Lido` que indica se um dado livro foi dido ou não, caso exista.

ii. Escreva uma função `livroAutor :: Biblio -> Autor -> [Livro]` que devolve a lista de livros do autor dado.

iii. Escreva uma função `naoLidos :: Biblio -> [Livro]` que devolve a lista de livros não lidos (considere a possibilidade desta lista estar ordenada pelo título do livro).

9 Classes

63. Considere o seguinte tipo de dados para representar fracções

```
data Frac = F Integer Integer
```

(a) Defina a função `normaliza :: Frac -> Frac`, que dada uma fracção calcula uma fracção equivalente, irredutível, e com o denominador positivo.

Por exemplo: `normaliza (F (-33) (-51)) = (F 11 17)` e `normaliza (F 50 (-5)) = (F (-10) 1)`. Relembre a função `mdc` que definiu no exercício 51c.

(b) Defina `Frac` como instância da classe `Eq`.

(c) Defina `Frac` como instância da classe `Ord`.

(d) Defina `Frac` como instância da classe `Show`, de forma a que cada fracção seja apresentada por *(numerador/denominador)*.

(e) Defina `Frac` como instância da classe `Num`. Relembre que a classe `Num` tem a seguinte definição

```
class (Eq a, Show a) => Num a where
  (+), (*), (-) :: a -> a -> a
  negate, abs, signum :: a -> a
  fromInteger :: Integer -> a
```

(f) Defina uma função que, dada uma fracção `f` e uma lista de fracções `l`, selecciona de `l` os elementos que são maiores do que o dobro de `f`.

64. Considere o seguinte tipo para representar expressões inteiras.

```
data ExpInt = Const Int
            | Simetrico ExpInt
            | Mais ExpInt ExpInt
            | Menos ExpInt ExpInt
            | Mult ExpInt ExpInt
```

Os termos deste tipo `ExpInt` podem ser vistos como árvores cujas folhas são inteiros e cujos nodos (não folhas) são operadores.

(a) Defina uma função `calcula :: ExpInt -> Int` que, dada uma destas expressões calcula o seu valor.

(b) Defina `ExpInt` como uma instância da classe `Show` de forma a que `show (Mais (Const 3) (Menos (Const 2)(Const 5)))` dê como resultado `"(3 + (2 - 5))"`.

(c) Defina uma outra função de conversão para strings `posfix :: ExpInt -> String` de forma a que quando aplicada à expressão acima dê como resultado `"3 2 5 - +"`.

(d) Defina `ExpInt` como uma instância da classe `Eq`.

(e) Defina `ExpInt` como instância desta classe `Num`.

65. Uma outra alternativa para representar expressões é como o somatório de parcelas em que cada parcela é o produto de constantes.

```
data ExpN = N [Parcela]
type Parcela = [Int]
```

(a) Defina uma função `calcN :: ExpN -> Int` de cálculo do valor de expressões deste tipo.

(b) Defina uma função de conversão `normaliza :: ExpInt -> ExpN`.

(c) Defina `ExpN` como instância da classe `Show`.

66. Relembre o exercício 55 sobre contas bancárias, com a seguinte declaração de tipos

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

(a) Defina `Data` como instância da classe `Ord`.

(b) Defina `Data` como instância da classe `Show`.

(c) Defina a função `ordena :: Extracto -> Extracto`, que transforma um extracto de modo a que a lista de movimentos apareça ordenada por ordem crescente de data.

(d) Defina `Extracto` como instância da classe `Show`, de forma a que a apresentação do extracto seja por ordem de data do movimento com o seguinte, e com o seguinte aspecto

```
Saldo anterior: 300
---------------------------------------
Data       Descricao   Credito   Debito
---------------------------------------
2010/4/5   DEPOSITO    2000
2010/8/10  COMPRA                37,5
2010/9/1   LEV                   60
2011/1/7   JUROS       100
2011/1/22  ANUIDADE              8
---------------------------------------
Saldo actual: 2294,5
```

(e) A função `dmaxDebito`, a seguir apresentada, calcula a data e o montante do maior débito de um extracto.

```
dmaxDebito :: Extracto -> Maybe (Data,Float)
dmaxDebito (Ext _ []) = Nothing
dmaxDebito (Ext s ((d,m,Debito x):t)) = maxdeb (d,x) (dmaxDebito (Ext s t))
dmaxDebito (Ext s (_:t)) = dmaxDebito (Ext s t)
```

Apresente a definição de `maxdeb` e indique claramente o seu tipo.

67. Uma possível generalização do tipo de dados apresentado no exercício 7, será considerar expressões cujas constantes são de um qualquer tipo numérico (i.e., da classe `Num`).

```
data Exp a = Const a
           | Simetrico (Exp a)
           | Mais (Exp a) (Exp a)
           | Menos (Exp a) (Exp a)
           | Mult (Exp a) (Exp a)
```

Declare `Exp` como instância da classe `Num`, completando a seguinte definição:

```
instance (Num a) => Num (Exp a) where
  .......
```

Note que, em rigor, deverá ainda definir o tipo `Exp a` como uma instância de `Show` e de `Eq`.

68. Considere o seguinte tipo:

```
data LTree a = Leaf a | Fork (LTree a) (LTree a)
```

(a) i. Defina uma instância da classe `Eq` para este tipo idêntica à que seria obtida com a directiva `deriving Eq`.

ii. Defina a função `mapLT :: (a -> b) -> LTree a -> LTree b` que aplica uma função a todas as folhas de uma árvore. Por exemplo,

```
> mapLT succ (Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2))
Fork (Fork (Leaf 1) (Leaf 2)) (Leaf 3)
```

(b) i. Defina uma instância da classe `Show` para este tipo por forma a obter o seguinte comportamento.

```
> Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2)
((0/\1)/\2)
```

ii. Defina a função `mktree :: Int -> a -> LTree a` que constroi uma árvore balanceada com um dado número de folhas todas iguais ao segundo argumento. Por exemplo,

```
> mktree 5 0
((0/\0)/\(0/\(0/\0)))
```

(c) i. Defina uma instância da classe `Eq` para este tipo que considere iguais quaisquer duas árvores com a mesma forma.

ii. Defina a função `build :: [a] -> LTree a` que constroi uma árvore com uma folha por cada elemento da lista argumento (que deve ser não vazia).

(d) i. Defina uma instância da classe `Show` para este tipo que apresente uma folha por cada linha, precedida de tantos pontos quanta a sua profundidade na árvore.

```
> Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2)
..0
..1
.2
```

Sugere-se que use uma função auxilar na definição da função `show`.

ii. Defina a função `cresce :: LTree a -> LTree a` que cresce uma árvore duplicando todas as suas folhas. Por exemplo,

```
> cresce (Fork (Fork (Leaf 0) (Leaf 1)) (Leaf 2))
...0
...0
...1
...1
..2
..2
```

10 Input/Output

69. A classe `Random` da biblioteca `System.Random` agrupa os tipos para os quais é possível gerar valores aleatórios. Algumas das funções declaradas nesta classe são:

- `randomIO :: Random a => IO a` que gera um valor aleatório do tipo `a`;
- `randomRIO :: Random a => (a,a) -> IO a` que gera um valor aleatório do tipo `a` dentro de uma determinada gama de valores.

Usando estas funções implemente os seguintes programas:

(a) `bingo :: IO ()` que sorteia os números para o jogo do bingo. Sempre que uma tecla é pressionada é apresentado um número aleatório entre 1 e 90. Obviamente, não podem ser apresentados números repetidos e o programa termina depois de gerados os 90 números diferentes.

(b) `mastermind :: IO ()` que implementa uma variante do jogo de descodificação de padrões *Mastermind*. O programa deve começar por gerar uma sequência secreta de 4 dígitos aleatórios que o jogador vai tentar descodificar. Sempre que o jogador introduz uma sequência de 4 dígitos, o programa responde com o número de dígitos com o valor correcto na posição correcta e com o número de dígitos com o valor correcto na posição errada. O jogo termina quando o jogador acertar na sequência de dígitos secreta.

70. Uma aposta do EuroMilhões corresponde à escolha de 5 *Números* e 2 *Estrelas*. Os *Números* são inteiros entre 1 e 50. As *Estrelas* são inteiros entre 1 e 9. Para modelar uma aposta destas definiu-se o seguinte tipo de dados:

```
data Aposta = Ap [Int] (Int,Int)
```

(a) Defina a função `valida :: Aposta -> Bool` que testa se uma dada aposta é válida (i.e. tem os 5 números e 2 estrelas, dentro dos valores aceites e não tem repetições).

(b) Defina a função `comuns :: Aposta -> Aposta -> (Int,Int)` que dada uma aposta e uma chave, calcula quantos *números* e quantas *estrelas* existem em comum nas duas apostas

(c) Use a função da alínea anterior para:

i. Definir `Aposta` como instância da classe `Eq`.

ii. Definir a função `premio :: Aposta -> Aposta -> Maybe Int` que dada uma aposta e a chave do concurso, indica qual o prémio que a aposta tem. Os prémios do EuroMilhões são:

Números	*Estrelas*	**Prémio**	*Números*	*Estrelas*	**Prémio**
5	2	**1**	3	2	**7**
5	1	**2**	2	2	**8**
5	0	**3**	3	1	**9**
4	2	**4**	3	0	**10**
4	1	**5**	1	2	**11**
4	0	**6**	2	1	**12**
			2	0	**13**

(d) Para permitir que um apostador possa jogar de forma interactiva:

i. Defina a função `leAposta :: IO Aposta` que lê do teclado uma aposta. Esta função deve garantir que a aposta produzida é válida.

ii. Defina a função `joga :: Aposta -> IO ()` que recebe a chave do concurso, lê uma aposta do teclado e imprime o prémio no ecrã.

(e) Defina a função `geraChave::IO Aposta`, que gera uma chave válida de forma aliatória.

(f) Pretende-se agora que o programa `main` permita jogar várias vezes e dê a possiblidade de simular um novo concurso (gerando uma nova chave). Complete o programa definindo a função `ciclo :: Aposta -> IO ()`.

```
main :: IO ()
main = do ch <- geraChave
          ciclo ch

menu :: IO String
menu = do { putStrLn menutxt
          ; putStr "Opcao: "
          ; c <- getLine
          ; return c
```

```
        }
  where menutxt = unlines ["",
                           "Apostar ........... 1",
                           "Gerar nova chave .. 2",
                           "",
                           "Sair .............. 0"]
```

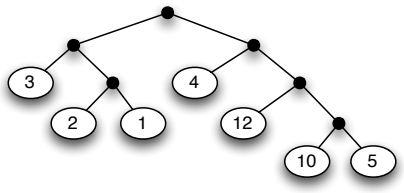
11 Problemas (parte 2)

71. Considere o seguinte tipo para representar árvores de folhas:

    ```
    data LTree a = Leaf a | Fork (LTree a) (LTree a)
    ```

    A cada folha de uma árvore pode ser associado o seu *"caminho"*, que não é mais do que uma lista de Booleanos (`False` esquerda e `True` direita). Por exemplo, na árvore da figura temos a seguinte correspondência entre nodos e caminhos:

    - O caminho `[False,False]` corresponde à folha 3.
    - O caminho `[False,True,True]` corresponde à folha 1.
    - O caminho `[True,True]` não corresponde a nenhuma folha da árvore.

    (a) Defina uma função `select :: LTree a -> [Bool] -> (Maybe a)` que determina a folha seleccionada por um caminho. Se o caminho não seleccionar nenhuma folha a função deve retornar `Nothing`.

    (b) Defina uma função `procura :: Eq a => LTree a -> a -> Maybe [Bool]` que, procura um elemento numa árvore e, em caso de sucesso calcula o caminho correspondente. Por exemplo, se `a` for a árvore representada na figura, `procura a 1 = Just [False,True,True]`.

    (c) Considere a seguinte função que lista as folhas de uma árvore, juntamente com a sua profundidade;

    ```
    travessia :: LTree a -> [(a,Int)]
    travessia (Leaf x) = [(x,0)]
    travessia (Fork e d) = map (\(x,n) -> (x,n+1)) (travessia e ++ travessia d)
    ```

    Defina uma função `build :: [(a,Int)] -> LTree a` inversa da anterior, i.e., tal que, `build (travessia a) = a` para toda a árvore `a`.

72. Para armazenar conjuntos de números inteiros, optou-se pelo uso de sequências de intervalos.

    ```
    type ConjInt = [Intervalo]
    type Intervalo = (Int,Int)
    ```

    Assim, por exemplo, o conjunto $\{1,2,3,4,7,8,19,21,22,23\}$ poderia ser representado por `[(1,4),(7,8),(19,19),(21,23)]`.

    (a) Defina a função `pertence :: Int -> ConjInt -> Bool` que testa se um dado inteiro pertence a um conjunto.

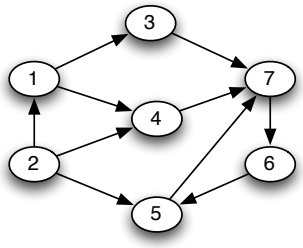
(b) Defina uma função `quantos :: ConjInt -> Int` que, dado um conjunto, dê como resultado o número de elementos desse conjunto. Por exemplo, `quantos [(1,4),(7,8),(19,19),(21,23)] = 10`.

(c) Defina uma função `elems :: ConjInt -> [Int]` que, dado um conjunto, dâ como resultado a lista dos elementos desse conjunto. Por exemplo, `elems [(1,4),(7,8),(19,19),(21,23)] = [1,2,3,4,7,8,19,21,22,23]`.

(d) Defina uma função `geraconj :: [Int] -> ConjInt` que recebe uma lista de inteiros, ordenada pordem crescente e sem repetições, e gera um conjunto. Por exemplo, `geraconj [1,2,3,4,7,8,19, 21,22,23] = [(1,4),(7,8),(19,19),(21,23)]`.

73. Uma relação binária entre elementos de um tipo `a` pode ser descrita como um conjunto (lista) de pares `[(a,a)]` ou, agregando todos os pares que têm a primeira componente em comum, por uma lista do tipo `[(a,[a])]`.

```
type RelP a = [(a,a)]
type RelL a = [(a,[a])]
```



Assim, a relação representada ao lado pode ser implementada por

- `[ (1,3), (1,4), (2,1), (2,4), 2,5), (3,7), (4,7), (5,7), (6,5), (7,6)] :: RelP Int`
- `[ (1,[3,4]), (2,[1,4,5]), (3,[7]), (4,[7]), (5,[7]), (6,[5]), (7,[6])] :: RelL Int`

(a) Considere a seguinte função de conversão entre representações:

```
converteLP :: RelL a -> RelP a
converteLP l = concat (map junta l)
    where junta (x,xs) = map (\y->(x,y)) xs
```

Defina a função de conversão `convertePL :: (Eq a) => RelP a -> RelL a` inversa da anterior, i.e., tal que `convertePL( converteLP l) = l` para todo `l`.

(b) Defina uma função `converso :: RelL a -> RelL a` que calcula a relação inversa de uma relação. Por exemplo, para o exemplo apresentado a relação inversa deve dar como resultado uma lista com os seguintes elementos:

`[(1,[2]), (3,[1]), (4,[1,2]), (5,[2,6]), (6,[7]), (7,[3,4,5])]`

74. Considere uma árvore genealógica ascendente (que relaciona uma pessoa com os seus pais) implementada no seguinte tipo de de dados:

```
data AG = Pessoa Nome Pai Mae
        | Desconhecida

type Pai = AG
type Mae = AG
type Nome = String
```

(a) Defina a função `nomesMF :: AG -> ([Nome],[Nome])` que, para a árvore genealógica de uma pessoa, calcula as listas com os nomes masculinos e os nomes femininos dos seus ascendentes.

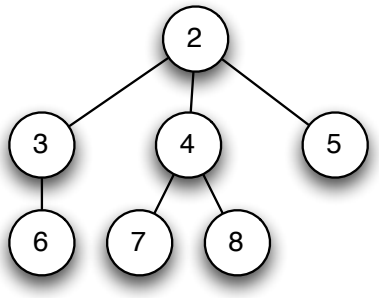
(b) Defina a função `avos :: Nome -> AG -> [Nome]` que recebe o nome de uma pessoa e uma árvore genealógica e dá a lista com os nomes dos avós dessa pessoa. Note que o nome da pessoa pode não estar na raiz da árvore. Assuma que não existem nomes repetidos na árvore.

(c) Defina a função `grau :: AG -> Nome -> Maybe Int` que dada uma árvore genealógica e o nome de uma pessoa, calcula o grau de parentesco entre essa pessoa e a pessoa que está na raiz da árvore. Assuma o grau de parentesco de pais é 1, de avós é 2, de bisavós é 3, etc. Assuma ainda que não existem nomes repetidos na árvore.

75. Considere a seguinte definição para representar árvores irregulares

```
data ArvIrr a = No a [ArvIrr a]
```

A expressão `No 2 [No 3 [No 6 []], No 4 [No 7 [], No 8 []], No 5 []]` representa a árvore



(a) Defina a função `maximo :: Ord a => ArvIrr a -> a` que calcula a maior elemento de uma destas árvores.

(b) A função `elems`, definida abaixo, lista os elementos de uma árvore, bem como o nível a que aparecem (segundo uma travessia pre-order).
Por exemplo, para a árvore apresentada, a função retorna a lista

```
[(2,1),(3,2),(6,3),(4,2),(7,3),(8,3),(5,2)]
```

```
elems a = elemsAux 1 a
   where elemsAux n (No x l) = (x,n):(concat (map (elemsAux (n+1)) l))
```

Defina a função `unElems :: [(a,Int)] -> ArvIrr a` inversa da anterior, no sentido em que, para toda a árvore `a`, `unElems (elems a) = a`.

76. A sequência 1, 11, 21, 1211, 111221, 312211, ... pode ser construída a partir do primeiro número – 1– seguido da descrição do número anterior:

- 11 pode ser lido como *um 1*
- 21 pode ser lido como *dois 1's*
- 1211 pode ser lido como *um 2, um 1*
- 111221 pode ser lido como *um 1, um 2's, dois 1*
- 312211 pode ser lido como *três 1's, dois 2's, um 1*

Assim, na sequência apresentada, o próximo elemento seria 13112221, correspondendo à descrição *um 3, um 1, dois 2's, dois 1's*.

(a) Escreva uma função `proximo :: Integer -> Integer` que dado um número da sequência, calcula o próximo (segundo o método descrito).

(b) Defina agora a função inversa `anterior :: Integer -> Integer`, que dado um elemento da sequência (com excepção do primeiro) dá o anterior. (Note que, à excepção do primeiro elemento da sequência, todos os elementos têm um número par de algarismos).

77. Para controlar um veículo tele-comandado, dispõe-se dos seguintes comandos:

RD – Rodar à direita: roda o veículo 90º no sentido dos ponteiros do relógio;

RE – Rodar à esquerda: roda o veículo 90º no sentido contrário ao dos ponteiros do relógio;

AV – Avançar: avança o veículo uma posição.

Para representar esses comandos definiu-se o tipo:

```
data Cmd = RD | RE | AV
     deriving (Eq,Show)
```

(a) Defina:

- O tipo `Pos` que represente a posição do veículo (i.e. coordenadas do ponto e orientação).
- A função `next :: Pos -> Cmd -> Pos`, que dada uma posição inícial e um comando calcula a posição do veículo após executar esse comando.

(b) Defina a função `percurso` que, dada uma posição inicial e uma lista de comandos, retorne a lista das posições percorridas pelo veículo (o percurso realizado).

(c) Defina um predicado que verifique, dadas as posições iniciais de dois veículos e as respectivas listas de comandos (uma por veículo), determine se ocorre uma colisão entre esses veículos (i.e. se existe um momento onde ambos os veículos ocupam uma posição no plano).

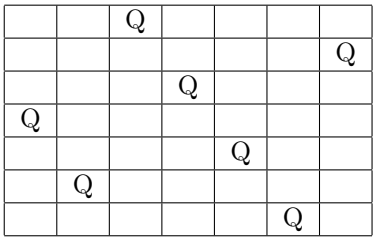
Considere para o efeito que após realizados todos os comandos contidos na lista, o veículo permanece "estacionado" na posição final.

(d) Pretende-se definir um simulador (`simula :: IO()`) para um veículo com a funcionalidade descrita nas alíneas anteriores. Para tal, considere que:

- o veículo desloca-se num tabuleiro de dimensões 10x10 (coordenadas (0,0) até (9,9));
- inicialmente são colocados obstáculos em 10 posições aleatórias do tabuleiro;
- a posição inicial do veículo é (0,0) com orientação vertical (virado para norte);
- após a inicialização do tabuleiro, o simulador entra num ciclo de interação com o utilizador solicitando um comando, e apresentando a nova posição/orientação do veículo;
- a interacção termina quando uma das seguintes condições se verificar:
  - o veículo sair fora dos limites do tabuleiro;
  - o veículo chocar com um dos obstáculos colocados no tabuleiro;
  - o veículo voltar a uma posição em que já tenha estado.

Relembre que para gerar um número aleatório compreendido entre `x` e `y` pode utilizar a função `randomRIO::(a,a) -> IO a`.

78. O problema das N raínhas consiste em colocar N rainhas num tabuleiro de xadrez com N linhas e N colunas, de tal forma que nenhuma rainha está ameaçada por outra. Note que uma rainha ameaça todas as posições que estão na mesma linha, na mesma coluna ou nas mesmas diagonais. No tabuleiro ao lado apresenta-se uma solução para este problema quando N é 7.



Uma forma de representar estas soluções é usando listas de strings. Por exemplo a lista `["Q_", "_Q"]` representa um tabuleiro de dimensão 2 com duas rainhas colocadas nos cantos superior esquerdo e inferior direito. Para o efeito fez-se a seguinte declaração de tipo

```
type Tabuleiro = [String]
```

(a) Defina uma função `rainhasOK` que testa se uma dada solução está correcta. Por exemplo, para o tabuleiro `["Q_", "_Q"]` a função deve dar `False` como resultado, uma vez que as raínhas se encontram na mesma diagonal.

(b) Defina a função `solucoes :: Int -> [Tabuleiro]` qua calcula todas as soluções possíveis para o problema com um dado número de raínhas. Por exemplo, para 4 raínhas a soluções serão

```
[["_Q__","___Q","Q___","__Q_"],["__Q_","Q___","___Q","_Q__"]]
```

79. Considere o seguinte tipo:

```
data List a b = Nil | ConsA a (List a b) | ConsB b (List a b)
```

(a) Escreva uma função `unzipAB :: List a b -> ([a],[b])` que produz as listas com todos os elementos dos tipos `a` e `b` contidos numa `List a b`

(b) Escreva uma função `sortA :: (Ord a) => List a b -> List a b` que ordena apenas os elementos de tipo `a` da lista, mantendo os elementos de tipo `b` inalterados.

80. Uma forma de representar relações binárias consiste em armazenar um conjunto (ou lista) de pares. Outras formas alternativas consistem em armazenar estes pares agrupados segundo a sua primeira componente. Desta forma, a relação representada como a seguinte lista de pares

```
[(1,'a'), (2,'b'), (5,'d'), (1,'e'), (6,'a'), (2,'f')]
```

seria representada por

- `[(1,['a','e']), (2,['b','f']), (5,['d']), (6,['a'])]`
- ou por `([1,2,5,6], g)` em que `g` é a função

```
g 1 = ['a','e']
g 2 = ['b','f']
g 5 = ['d']
g 6 = ['a']
```

Considere então os seguintes tipos correspondentes a estas três representações:

```
type Rel1 a b = [(a,b)]
type Rel2 a b = [(a,[b])]
type Rel3 a b = ([a], a -> [b])
```

(a) Defina as funções de conversão entre as várias representações:

```
rel12 :: (Eq a) => (Rel1 a b) -> Rel2 a b
rel23 :: (Eq a) => (Rel2 a b) -> Rel3 a b
rel31 :: (Rel3 a b) -> Rel1 a b
```

(b) Defina uma função `compoe :: (Eq c) => (Rel2 a c) -> (Rel2 c b) -> (Rel2 a b)` que calcula a composição de relações.

A composição de relações é semelhante à composição de funções: sempre que a está relacionado com b na primeira relação e b está relacionado com c na segunda, então a está relacionado com c na composição das relações.

(c) Defina uma função `inverte :: (Eq b) => (Rel2 a b) -> (Rel2 b a)` que calcula a inversa de uma relação.

Relembre que se a está relacionado com b numa relação, então b está relacionado com a na relação inversa.