Programação Funcional

2012/13

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. 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.

9. 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

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

11. 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.

12. 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

13. 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.

14. 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 8, 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).

15. 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)]`

16. 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.

4 Funções recursivas que devolvem "tuplos"

17. 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.

18. 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.

19. 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`).

20. 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

21. 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]]`

22. 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

23. 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.

24. 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)]`.

25. 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`.

26. 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

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

27. 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`.

28. 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`.

29. 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],[]]`.

7 Problemas numéricos

30. 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.

31. 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

32. 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.

33. 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)
```

34. 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 polonómio 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

35. 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.

36. 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.

37. 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).

38. 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, da 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.

39. 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.

9 Classes

40. 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 33c.

(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`.

41. 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`.

42. 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`.

43. Relembre o exercício 37 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 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 ((_,Debito,d,m):t) = maxdeb (d,m) (dmaxDebito t)
dmaxDebito (h:t) = dmaxDebito t
dmaxDebito [] = Nothing
```

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

44. 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`.

10 Input/Output

45. 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.

46. 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"]
```