DEPARTAMENTO DE INFORMÁTICA
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

PRINCÍPIOS DE PROGRAMAÇÃO

1º Exame 19 de Janeiro de 2011 **Cotação**: 14 valores
Duração: 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*1+1+1+1 valores*] Para cada alínea abaixo, responda e justifique:
   a) Qual o valor da expressão `foldl (+) 10 [1..5]`?
   b) Qual o tipo mais geral da expressão `\(x,y) -> x + fst y`?
   c) Defina, em Haskell, um tipo que represente uma árvore ternária.
   d) Qual a diferença entre *lazy evaluation* e *strict evaluation*?

2) [*2+2+2 valores*] Defina as seguintes funções (pode utilizar as funções do `Prelude`):
   a) `getCell :: [[Int]] -> Int -> Int -> Int`, onde `getCell m i j` é o valor da i-ésima linha e da j-ésima coluna (os índices começam em zero) da matriz `m` dada (a matriz dada é uma lista de linhas, e todas as linhas têm o mesmo tamanho). A função deve devolver `-1` se `i` ou `j` não representarem um índice válido.
   b) `countP :: [a] -> (a -> Bool) -> Int`, que é o número de elementos da lista dada que satisfazem o predicado dado. Por exemplo, a aplicação `countP [1,-2,0,-1,5] (>0)` é `2`. Defina a função com recursão terminal (se não conseguir, defina-a de outra forma mas terá menor cotação).
   c) `sumGT :: (Ord a, Num a) => [a] -> a`, que é a soma dos valores positivos da lista dada. Deve implementar esta função nas seguintes versões: (i) usar `sum` e listas por compreensão, (ii) usando `sum, filter` e o operador de composição (iii) usando `foldr`.

3) [*2 valores*] Verifique que a propriedade `sum (xs++ys) = sum xs + sum ys` é satisfeita pelas funções `sum` e `(++)` para todas as listas finitas `xs` e `ys`.

```
sum []     = 0                  (sum1)
sum (x:xs) = x + sum xs         (sum2)

  []   ++ ys = ys               (++1)
(x:xs) ++ ys = x:(xs++ys)       (++2)
```

4) [*2 valores*] Defina a classe `ForEachInt a` que descreve a travessia para estruturas de dados que armazenam inteiros e que contempla as seguintes funções: `felength` (o número de elementos na estrutura), `fehead` (o primeiro inteiro na estrutura), `fetail` (a estrutura sem o primeiro inteiro), `fe2list` (devolve os inteiros da estrutura numa lista), `feget` (o i-ésimo inteiro de acordo com `felist`). Dê uma definição por defeito da função `feget`.

Pergunta complementar do projecto.

Esta pergunta deve ser respondida apenas para quem não obteve o valor mínimo requerido no projecto após ponderação do teste individual.

Defina a função `bestScore :: Int -> Int -> Int -> Score` onde `bestScore u1 u2 s` é o melhor confronto, da perspectiva do primeiro jogador, dado pela matriz de confrontos.

Define-se o melhor confronto para o primeiro jogador como o confronto que possui o maior valor obtido pela subtracção das suas vitórias pelas suas derrotas. Em caso de haver mais do que um melhor confronto, devolver qualquer um deles.

Lembrar que a função `matrixScores :: Int -> Int -> Int -> [[Score]]` calcula a matriz de confrontos.

DEPARTAMENTO DE INFORMÁTICA
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

PRINCÍPIOS DE PROGRAMAÇÃO

1º Exame 2 de Fevereiro de 2011 **Cotação**: 14 valores
Duração: 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*1+1+1+1 valores*] Para cada alínea abaixo, responda e justifique:
   a) Qual o valor da expressão `map ((*5).(+4)) [1..3]`?
   b) Qual o tipo mais geral da expressão `\f g -> f (g 1 == "ab")`?
   c) Seja o tipo `data Tree = Leaf Int | Node Tree Int Tree`. Desenhe a árvore que corresponde ao valor
   ```
   Node (Leaf 1) 2 (Node (Leaf 3) 4 (Leaf 5))
   ```
   d) Mostre um exemplo onde a avaliação preguiçosa seja mais eficiente que a avaliação gananciosa. Não use listas infinitas.

2) [*2+2+2 valores*] Defina as seguintes funções (pode utilizar as funções do `Prelude`):
   a) `swapPairs :: [a] -> [a]`, que troca um elemento da lista com o próximo, repetindo isso de par em par. Se a lista contiver um número ímpar de elementos, o último elemento não é modificado. Exemplo: `swapPairs [1,2,3,4,5]` resulta em `[2,1,4,3,5]`.
   b) Usando a definição de árvore da alínea 1c, defina a função `nSatisfy :: (Int->Bool) -> Tree -> Int` que corresponde ao número de elementos na árvore dada que satisfazem o predicado dado.
   c) `iterProgressive :: (Int->Int) -> [Int]`, que dada a função `f`, é a lista infinita `[f 1, f`^{2} `2, f`^{3} `3,...]`. A notação f^n representa a composição da função f, n vezes. Por exemplo, `take 6 (iterProgressive (+1))` resultaria na lista `[2,4,6,8,10,12]`.

3) [*2 valores*] Verifique que a propriedade `reverse (xs++ys) = reverse ys ++ reverse xs` é satisfeita pelas funções `reverse` e `(++)` para todas as listas finitas `xs` e `ys`.
   ```
   reverse []     = []                    (rev1)
   reverse (x:xs) = reverse xs ++ [x]     (rev2)

     []   ++ ys = ys                      (++1)
   (x:xs) ++ ys = x:(xs++ys)              (++2)
   ```
   Nota: verifique, como resultado auxiliar, a propriedade `xs ++ [] = xs`.

4) [*2 valores*] Utilizando as funções `length` e `(!!)`, escreva uma especificação da função do `Prelude`, `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]`.

Pergunta complementar do projecto.

Esta pergunta deve ser respondida apenas para quem não obteve o valor mínimo requerido no projecto após ponderação do teste individual.

Defina a função `bestOf3 :: Strategy -> Strategy -> Strategy -> Strategy` que devolve a melhor das três estratégias dadas (ou uma das melhores, se houver empates).

Lembrar que existe a função `compareSt :: Strategy -> Strategy -> Score` que devolve o confronto entre duas estratégias.

DEPARTAMENTO DE INFORMÁTICA
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

PRINCÍPIOS DE PROGRAMAÇÃO

1º Exame 18 de Janeiro de 2012 **Cotação**: 16 valores
Duração: 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*1.5+1.5+1.5+1.5 valores*] Para cada alínea abaixo, responda e justifique:

a) Qual o valor da expressão

```
foldr (*) 4 (filter (>0) [-1,1,-2,2,-3,3]) ?
```

b) Qual o tipo mais geral da expressão

```
\f g x -> g (f (x `mod` 2) == "sim") ?
```

c) Seja o tipo

```
data Tree = Leaf | Node Tree Int Tree
```

Desenhe uma árvore qualquer que possa ser representada pelo tipo `Tree`, seja árvore de pesquisa e contenha os valores inteiros 1, 2, 3, 4 e 5.

Indique ainda o valor do tipo `Tree` que representa a árvore que desenhou.

Uma árvore binária é *de pesquisa* se, para qualquer nó, as suas sub-árvores esquerda e direita são árvores de pesquisa, quaisquer nós na sub-árvore esquerda têm valor inferior ao do nó considerado, e quaisquer nós na sub-árvore direita têm valor superior ao do nó considerado

d) Mostre um exemplo de uma expressão cuja avaliação seria impossível na prática se se usasse avaliação gananciosa, mas cuja avaliação se torna possível usando avaliação preguiçosa.

2) [*3+3 valores*] Defina as seguintes funções (pode utilizar as funções do `Prelude`).

a) Considere a função `pairs` que forma uma lista de pares a partir de uma lista,

```
pairs :: [a] -> [(a,a)]
pairs xs = zip xs (tail xs)
```

Usando `pairs` e recorrendo a uma definição por compreensão, defina a função

```
unsorted :: Ord a => [a] -> Bool
```

que devolve `True` caso haja pelo menos um par de elementos adjacentes na lista argumento que não estejam por ordem crescente, `False` caso contrário.

b) Usando a definição de árvore da alínea 1c, defina a função

```
totalNumPares :: Tree -> Int
```

que obtém o número de elementos armazenados numa árvore que sejam números pares.

3) [*2 valores*] Verifique que a propriedade

```
reverse (reverse xs) = xs
```

é satisfeita pelas funções `reverse` e `(++)` para todas as listas finitas `xs`.

```
reverse []     = []                    (rev1)
reverse (x:xs) = reverse xs ++ [x]     (rev2)

  []   ++ ys = ys                      (++1)
(x:xs) ++ ys = x:(xs++ys)              (++2)
```

Nota: pode usar, considerando como anteriormente provada, a propriedade

`reverse (xs++ys) = reverse ys ++ reverse xs.`

4) [*2 valores*] Defina o tipo `Pixel` com a representação de dados que achar conveniente supondo que a informação a guardar consiste na posição do pixel no ecrã e num número correspondente ao nível de cinzento na respectiva posição. Pode supor que o nível de cinzento está entre 0 ("preto") e 255 ("branco").

Torne `Pixel` uma instância das classes `Eq` e `Show` seguindo o critério de que dois pixeis são iguais se tiverem o mesmo nível de cinzento, independentemente da posição. Para a função `show` é preciso mostrar toda a informação relevante do pixel.

DEPARTAMENTO DE INFORMÁTICA
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

PRINCÍPIOS DE PROGRAMAÇÃO

2º Exame 1 de Fevereiro de 2012 **Cotação**: 16 valores **Duração:** 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*1.5+1.5+1.5 valores*] Para cada alínea abaixo, responda e justifique:

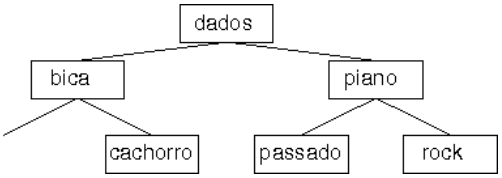
a) Qual o tipo mais geral da expressão

```
foldr (*) 4 (filter (>0) [-1,1,-2,2,-3,3])
```
?

b) Seja o tipo parametrizado

```
data Tree a = EmptyLeaf | Leaf a | Node (Tree a) a (Tree a)
```

Respeitando a sintaxe de `Tree a`, indique o valor do tipo `Tree String` que representa a árvore binária da figura abaixo.



c) Indique, justificando, quais das seguintes expressões podem ser avaliadas na prática:

1ª: `take 3 [1..]`

2ª: `drop 3 [1..]`

3ª: `take 3 ([1..] ++ [1..])`

4ª: `take 3 ([1..] ++ drop 3 [1..])`

2) [*3+2 valores*] Defina as seguintes funções (pode utilizar as funções do `Prelude`).

a) Usando a definição de árvore parametrizada da alínea 1b, defina a função

```
find :: Ord a => Tree a -> a -> Bool
```

que pesquisa um elemento do tipo `a` dentro de uma `Tree a`, devolvendo `True` caso o elemento ocorra, `False` caso contrário.

Assuma como pré-condição que a árvore é de pesquisa.

Uma árvore binária é *de pesquisa* se, para qualquer nó, as suas sub-árvores esquerda e direita são árvores de pesquisa, quaisquer nós não-vazios na sub-árvore esquerda têm valor inferior ao do nó considerado, e quaisquer nós não-vazios na sub-árvore direita têm valor superior ao do nó considerado.

b) Ainda com a definição de árvore parametrizada da alínea 1b, *complete* a definição da seguinte função

```
tree2list :: Ord a => Tree a -> [a]
tree2list EmptyLeaf = []
tree2list (Leaf x)  = [x]
```

de modo a que a função permita obter uma lista ordenada dos elementos de uma árvore de pesquisa.

3) [*1.5+1.5 valores*] Para o programa abaixo, é útil relembrar os tipos de duas funções,

```
writeFile  :: FilePath -> String -> IO ()
readFile   :: FilePath -> IO String
```

a) No programa seguinte, descreva a acção da função `main`.

```
mostraFich :: FilePath -> IO ()
mostraFich nome = do
                  minhaString <- readFile nome
                  putStr minhaString


guardaNums :: Int -> IO ()
guardaNums n =
           writeFile "nums.txt" (numToText [x*x | x <- [1..n]])
           where
             numToText = concat . map ( (++"\n").show )


main :: IO ()
main = do
        guardaNums 10
        putStr "Deseja ver o ficheiro? (sim/nao): "
        resposta <- getLine
        if (resposta == "sim") then mostraFich "nums.txt"
                               else return ()
```

b) Defina uma função

```
multiplo :: IO ()
```

que peça dois números inteiros ao utilizador, um em cada linha, e indique ao utilizador se o segundo é (ou não) múltiplo do primeiro.

4) [*1+1+1.5 valores*] Na definição do tipo abaixo, tenha em atenção a necessidade de definir as operações solicitadas; escolha a estrutura de dados adequada.

a) Defina um tipo `Vector` que consiga representar adequadamente vectores de números inteiros. Cada vector tem dimensão (finita) que é arbitrária, mas que está bem definida para esse vector. Para cada vector, deve ser possível conhecer a sua dimensão e as suas componentes.

Faça o necessário para que `Vector` seja instância da classe `Eq`.

b) Defina uma função

```
origem :: Int -> Vector
```

que devolva um vector com todas as componentes iguais a 0 e dimensão dada pelo argumento inteiro.

c) Defina uma função

```
somaVec :: Vector -> Vector -> Vector
```

que efectue a soma de dois vectores. Assuma como pré-condição que os vectores argumento têm a mesma dimensão.

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

Princípios de Programação

Exame de 1ª época — 17 de Janeiro de 2013

Duração: 2h30m — Cotação: 16 valores

Identifique completamente o seu caderno de exame, escrevendo os seus número e nome em letra de imprensa bem legível.

Leia o enunciado com atenção e justifique *sempre* as suas respostas.

Pode usar funções do Prelude, excepto aquelas que sejam explicitamente proibidas.

Parte 1

[$1+1+1+1$ *valores*]

(a) Qual é o valor da expressão `foldr (/) 1 [12,2,3]` ?

(b) Quais são as vantagens da avaliação preguiçosa no Haskell?

(c) Qual é o tipo mais geral da função `flip` definida por `flip f x y = f y x` ?

(d) Qual é o tipo mais geral da expressão `\f g x -> g (f (x `mod` 2) == "exame de PP")` ?

Parte 2

Defina as funções nas questões 2.1 a 2.3.

Questão 2.1 [2 *valores*]

`isSuffix :: Eq a => [a] -> [a] -> Bool`, que verifica se a segunda lista é sufixo da primeira. Considere que as listas são finitas. Por exemplo, `isSuffix [1..9] [8,9]` devolve `True`. Sugestão facultativa: pode usar como auxiliar a função `reverse`, que inverte a ordem dos elementos de uma lista finita.
Uma lista B (com n elementos) é sufixo de uma lista A se a lista constituída pelos n últimos elementos de A for igual a B.

Questão 2.2 [2 *valores*]

`myMap :: (a -> b) -> [a] -> [b]`, que se comporta exactamente como a função `map` do Prelude, ou seja, a avaliação de `myMap g xs` resulta numa lista em que a função `g` é aplicada a cada elemento de `xs`, alterando assim (em geral) o valor em cada posição. Deve definir `myMap` usando a função `foldl`. Se não conseguir responder usando `foldl`, pode usar a `foldr` (tendo menor cotação) ou apresentar qualquer outra resolução (com cotação ainda menor). Não pode usar a função `map` do Prelude nesta resposta. **Esta questão continua na próxima página.**

Eis duas definições possíveis para `foldl` e `foldr`, respectivamente:

```
foldl f v [] = v
foldl f v (x:xs) = foldl f (f v x) xs

foldr f v [] = []
foldr f v (x:xs) = f x (foldr f v xs)
```

Questão 2.3 [2 *valores*]

`scalarProdSquare :: [Int] -> [Int] -> Int`, que, dadas duas listas de inteiros, devolve a soma *dos quadrados* dos produtos entre os respectivos i-ésimos elementos. Se uma lista contiver mais elementos que a outra, esses elementos extra são ignorados. Por exemplo, a avaliação de `scalarProdSquare [1,2] [3..5]` resulta em `73`, i.e., o resultado de `(1*3)^2+(2*4)^2`. A sua resolução deve estar na forma de *recursão terminal*. Deve demonstrar que a recursão é terminal. Se não conseguir responder usando recursão terminal, indique qualquer outra resolução que funcione — mas terá cotação inferior.

Parte 3

Questão única [2 *valores*]

Prove, por indução sobre a estrutura de lista, que a seguinte propriedade é satisfeita pelas funções `length` e `(++)` para todas as listas finitas `xs` e `ys`:

```
length (xs ++ ys) = length xs + length ys
```

Use as seguintes definições para `length` e `(++)`:

```
length []     = 0                (length_1)
length (_:xs) = 1 + length xs    (length_2)

[]     ++ ys = ys                (++_1)
(x:xs) ++ ys = x : (xs ++ ys)    (++_2)
```

Parte 4

Questão única [4 *valores*]

Defina o tipo `Rectangle` com a representação de dados que achar conveniente, respeitando as condições: todos os tipos numéricos usados devem ser de vírgula flutuante; é necessário conhecer a posição do rectângulo no plano, os comprimentos dos seus lados, e a orientação do mesmo rectângulo em relação a uma linha de base arbitrária — por exemplo uma linha horizontal. Torne este tipo instância das classes `Eq` e `Show`. Para a classe `Eq`, deve-se adoptar o critério de que dois rectângulos são equivalentes se se conseguir encontrar um lado em cada rectângulo, tais que os respectivos comprimentos tenham diferença inferior a 0.01, e verificando ainda que os dois comprimentos de lados que sobram (um em cada rectângulo) também têm uma diferença inferior a 0.01. Sugestão: a função `abs` permite determinar o valor absoluto de um número. Quanto à função `show`, esta deve mostrar toda a informação relevante do rectângulo. Documente cuidadosamente a representação de dados, justificando as suas escolhas.

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

Princípios de Programação

Exame de 2ª época — 30 de Janeiro de 2013

Duração: 2h30m — Cotação: 16 valores

Identifique completamente o seu caderno de exame, escrevendo os seus número e nome em letra de imprensa bem legível.

Leia o enunciado com atenção e justifique *sempre* as suas respostas.

Pode usar funções do Prelude.

Parte 1

[$1.5 + 1 + 1.5$ *valores*]

(a) Qual é o valor da expressão `foldl (/) 60 [2,3,4]`? Apresente todos os cálculos.

(b) Explique porque é que uma estratégia de avaliação estrita impediria a avaliação de `take 10 ([1..] ++ drop 10 [1..])`.

(c) Qual é o tipo mais geral da expressão `\xs ys f -> f (length xs == length ys) == 1`?

Parte 2

Questão 2.1 [2 *valores*]

Eis duas variantes de uma função que calcula os termos da sucessão de Fibonacci.

```
fib :: Integer -> Integer
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)

fib' :: Integer -> Integer
fib' 0 = 1
fib' 1 = 1
fib' n = fibAux (1,1,n)
  where
    fibAux (x,_,0) = x
    fibAux (x,y,n) = fibAux (y,x+y,n-1)
```

(a) Qual é a complexidade espacial de `fib` expressa na notação *O*-grande? Justifique.

(b) Qual é a complexidade temporal de `fib` expressa na notação *O*-grande? Justifique.

(c) Qual é a complexidade temporal de `fib'` expressa na notação *O*-grande? Justifique.

Questão 2.2 [2 *valores*]

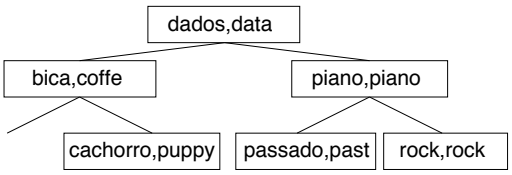
Defina a função `aplicaTodas :: [a -> a] -> a -> a`, que aplica todas as funções da lista no primeiro argumento ao segundo argumento, da direita para a esquerda.
Note que `aplicaTodas []` se comporta como a função identidade.
Exemplo: `aplicaTodas [(*3),(+1),\x->x+5] 2` resulta em `24`.
Se definir `aplicaTodas` usando a função `foldr`, terá a cotação máxima de 2.0 valores.
Se não conseguir responder usando `foldr`, pode apresentar qualquer outra resolução, com cotação máxima de 1.6 valores.

Questão 2.3 [2 *valores*]

Considere a implementação do TDA Dictionary sobre uma estrutura de árvore binária de pesquisa, como esboçado no ficheiro abaixo. Pretende-se aqui concretizar com dicionários português-inglês.

```
module Dictionary (Dictionary, emptyDic, insertEntry, translate) where

data Tree a = EmptyLeaf | Leaf a | Node (Tree a) a (Tree a)

data Dictionary = Dic (Tree (String,String))

emptyDic :: Dictionary
emptyDic = Dic EmptyLeaf

-- insere uma entrada no dicionário,
-- mantendo a estrutura de árvore de pesquisa
insertEntry :: (String,String) -> Dictionary -> Dictionary
... -- NÃO É NECESSÁRIO COMPLETAR NESTE EXAME

translate :: String -> Dictionary -> String
translate pt (Dic someTree) = translateTree pt someTree
  where
    translateTree _ EmptyLeaf = "not found"
    translateTree pt (Leaf (pt',en)) = if pt == pt' then en
                                                    else "not found"
... -- COMPLETAR
```

Um exemplo de dicionário:



Complete a definição recursiva da função `translate`, tal que, por exemplo, a tradução de `"cachorro"` usando o dicionário ilustrado resulta em `"puppy"`.

Parte 3

Questão única [2 *valores*]

Defina uma função `testStringPrefix :: IO ()` que peça duas strings ao utilizador, uma em cada linha, e indique ao utilizador se a primeira string é prefixo da segunda.

Uma lista A (com n elementos) é prefixo de uma lista B se a lista constituída pelos n primeiros elementos de B for igual a A.

As seguintes funções podem ser úteis:

```
getLine :: IO String
putStr :: String -> IO ()
putStrLn :: String -> IO ()
```

Parte 4

Questão única [4 *valores*]

Relembre o tipo `Tree a` definido na página 2:

```
data Tree a = EmptyLeaf | Leaf a | Node (Tree a) a (Tree a)
```

Complete as seguintes declarações de instância.

```
instance Show a => Show (Tree a) where
  ...
instance Eq a => Eq (Tree a) where
  ...
```

Departamento de Informática **Princípios Programação 2013–14**
Faculdade de Ciências **Exame 1**
Universidade de Lisboa **20 janeiro 2014**

Duração: 2h30m. Sem consulta. Inicie cada grupo numa nova página. Junte uma assinatura a cada função que escrever. Pode usar qualquer função constante no prelude.

Grupo 1. [Tipos e funções recursivas; 2 valores]

a) Qual o tipo da expressão **curry id**? E o tipo de **curry** (**curry id**)? Justifique. (Lembre-se que **id** tem o tipo a −> a).

```
curry :: ((a,b) -> c) -> (a -> b -> c)
curry g x y = g (x,y)
```

b) Escreva uma função *recursiva* conta :: **String** −> (**Int**, **Int**, **Int**) que, dada uma cadeia de carateres, devolva o número de caracteres, o número de palavras e o número de linhas na cadeia. O fim de linha é assinalado pela marca de fim de linha, as palavras são separadas por um número variável de espaços ou caracteres de tabulação.

Grupo 2. [Construção de tipos; 2 valores]

Para interpretar programas imperativos é necessário guardar o valor das diversas variáveis que ocorrem nos programas. A estrutura de dados Armazém é utilizada para este efeito. Pretende-se implementar as seguintes funções,

```
type Variável = String
inicial :: Armazém
atualizar :: Armazém -> Variável -> Integer -> Armazém
valor :: Armazém -> Variável -> Integer
```

onde inicial representa um armazém vazio (sem variável alguma); atualizar junta uma nova variável ao armazém ou altera o valor da variável se esta já estiver no armazém; e a operação valor devolve o valor associado a uma dada variável (devolve 0 caso a variável não se encontre no armazém).

a) Defina um tipo de dados Armazém para descrever armazéns de variáveis. Sugestão: utilize uma lista de pares variável-inteiro.

b) Implemente as três funções descritas acima.

Grupo 3. [Módulos e instâncias de classes de tipos; 2 valores]

a) Escreva o cabeçalho de um módulo que exporte o tipo de dados Armazém e as três funções, mas não o(s) construtor(es) do tipo de dados Armazém.

b) Torne o tipo de dados Armazém instância da classe de tipos **Eq**, de modo a que dois armazéns sejam iguais quando associam inteiros iguais a variáveis iguais.

Grupo 4. [QuickCheck; 2 valores]

a) Escreva uma propriedade QuickCheck que relacione a operação valor com o armazém inicial : o valor de qualquer variável no armazém inicial é 0.

b) Escreva uma propriedade QuickCheck que relacione a operação valor com o armazém obtido pela operação atualizar : dado um armazém a, se efectuarmos a atualização de uma variável v com um valor n e depois pedirmos o valor da mesma variável obtemos o número com que a variável foi atualizada (isto é n). Se pelo contrário, pedirmos o valor de uma outra variável v' (diferente de v) então obtemos o valor da variável no armazém dado (isto é, a).

c) Torne o tipo que definiu no grupo **2** alínea **a)** numa instância da classe Arbitrary. Não se esqueça que arbitrary deve ser um valor bem formado do tipo Armazém.

Grupo 5. [Entrada/saída e mónades; 2 valores]

a) Escreva um programa que repetidamente lê números inteiros até encontrar o número 0 e que escreva uma versão ordenada dos números lidos.

b) A função **map** pode ser escrita utilizando listas em compreensão da seguinte forma.

```
map' f xs = [f x | x <- xs]
```

Escreva a mesma função usando a notação **do**.

Departamento de Informática **Princípios Programação 2013–14**
Faculdade de Ciências **Exame 2**
Universidade de Lisboa **4 fevereiro 2014**

Duração: 2h30m. Sem consulta. Inicie cada grupo numa nova página. Junte uma assinatura a cada função que escrever. Pode usar qualquer função constante no prelude.

Grupo 1. [Tipos e funções recursivas; 2 valores]

a) Considere a seguinte função.

```
duasVezes f x = f (f x)
```

É possível aplicar a função duasVezes à função **elem** e à lista [[5],[5]] ? Justifique. Qual o tipo mais geral da função duasVezes? Justifique.

b) Defina a função calcula :: **String** −> **Int** −> [**Int**] que, dada uma cadeia de caracteres (chamados comandos) e um inteiro (chamado raiz), calcula uma lista de números, lendo a cadeia caracter a caracter, e efectuando as operações abaixo descritas. Se ler:

- '+' aumenta a raiz em uma unidade e junta-a à lista, o valor calculado constitui a nova raiz;
- '-' diminui a raiz em uma unidade e junta-a à lista, o valor calculado constitui a nova raiz;
- '[' guarda a raiz actual;
- ']' repõe o valor da raiz guardada no último '[' .

Por exemplo, a expressão calcula "++[−[++]−−]+" 5 deve produzir a lista [6,7,6,7,8,5,4,8] como resultado.

Grupo 2. [Construção de novos tipos; 3 valores]

Conjunto é uma estrutura de dados utilizada com bastante frequência. Uma implementação eficiente utiliza árvores de pesquisa. Considere a seguinte implementação de conjuntos de elementos equipados com uma relação de ordem.

```
data Conjunto a = Vazio | No (Conjunto a) a (Conjunto a)
```

a) Escreva uma expressão vazio que denote um conjunto vazio.

b) Escreva uma função singular que devolva um conjunto singular, dado o seu elemento.

c) Escreva uma função inserir que insira um elemento na árvore de pesquisa que representa um dado conjunto.

d) Utilizando a seguinte função,

```
fold :: b -> (b -> a -> b -> b) -> Conjunto a -> b
fold x _ Vazio       = x
fold x f (No e y d) = f (fold x f e) y (fold x f d)
```

escreva função a elementos que devolve uma lista ordenada de todos os elementos constantes num dado conjunto.

Grupo 3. [Módulos e instâncias de classes de tipos; 2 valores]

Para a resolução deste grupo pode utilizar as funções desenvolvidas no grupo anterior.

a) Escreva o cabeçalho de um módulo que exporte o tipo de dados Conjunto e as três funções (vazio, singular, inserir e elementos), mas não o(s) construtor(es) do tipo de dados Conjunto.

b) Torne o tipo de dados Conjunto instância da classe de tipos **Eq**, de modo a que dois conjuntos sejam iguais quando contenham exactamente os mesmos elementos.

c) Torne o tipo de dados Conjunto instância da classe de tipos **Show**, de modo a apresentar um conjunto por extensão, como é prática comum em matemática: uma lista de elementos, separados por vírgulas e colocados entre chavetas. Exemplos {}, {1}, {1,2,3}.

Grupo 4. [Raciocinio sobre programas; 2 valores]

Relembre as seguintes funções constantes do prelude.

```
drop :: Int -> [a] -> [a]
drop n xs     | n <= 0 = xs                    (drop/1)
drop _ []              = []                    (drop/2)
drop n (_:xs)          = drop (n-1) xs         (drop/3)
```

```
id :: a -> a
id x = x                                                    (id/1)

(.) :: (b -> c) -> (a -> b) -> a -> c
(.) f g = \x -> f (g x)                                     (./1)
```

O *princípio da extensionalidade* diz que, para mostrar que duas funções f e g são iguais, mostramos que f x = g x para todos os elementos x no domínio das funções.

a) Mostre que a seguinte equação é válida

```
drop (−500) = id
```

mostrando a validade da equação **drop** (−500) xs = **id** xs para todas as listas finitas xs.

b) Mostre que

```
drop m . drop n = drop (m + n)
```

Grupo 5. [Entrada/saída e mónades; 2 valores]

a) Escreva uma função acumula :: [**IO** a] −> **IO** [a] que executa uma sequência de interações, acumulando o resultado numa lista. Por exemplo:

```
*Main> acumula [getLine, getLine]
olá
mundo!
["olá","mundo!"]
```

b) Descreva o tipo e o valor de cada uma das duas expressões abaixo.

```
Just 3 >>= (\x -> Nothing >>= (\y -> Just (show x ++ y)))
return 3 >>= (\x -> Just "!" >>= (\y -> Just (show x ++ y)))
```

Princípios de Programação

DI / FCUL, Exame de 1ª época

19 de janeiro de 2015

Duração: 2h30
Cotação: 14 valores, $\star$ = 0.5 valores
Resolva cada grupo em folhas diferentes

1. Grupo 1 (3 valores)

1.[$\star$] Quais são as vantagens e as desvantagens da avaliação preguiçosa em Haskell?

2.[$\star\star$] Qual o valor da expressão: `foldl (*) 4 (filter (>0) [-1,1,-2,2,-3,3])` ?

3.[$\star\star$] Qual o tipo mais geral da expressão : `\f g x -> g (f (x `mod` 3) == "barbatana")` ?

4.[$\star$] Na sua opinião, o mecanismo de avaliação preguiçosa permite avaliar a expressão `take 3 (reverse [1..])` ? Na afirmativa mostre como, detalhando cada passo. Em qualquer caso justifique a sua resposta.

2. Grupo 2 (4.5 valores)

1.[$\star\star\star$] Defina uma função `getOddIdx` recursiva que recebe uma lista e retorna uma lista composta dos elementos que estão nas posições ímpares na lista original.

2.[$\star\star\star$] Defina a função `sumImp :: (Num a) => [a] -> a`, que é a soma dos valores ímpares da lista dada. Deve implementar esta função usando a função `foldr`.

3.[$\star\star\star$] Escreva uma função recursiva `slice :: [a] -> Int -> [[a]]` que recebe uma lista e um número inteiro e retorna uma lista de listas onde os elementos estão agrupados em lista de comprimento igual ao segundo argumento. **Usar recursão terminal** (caso contrário terá um desconto de 50% nesta pergunta).Exemplo : `slice [3,4,1,6,7,1,9] 3` dá `[[3,4,1],[6,7,1],[9]]`

3. Grupo 3 (2 valores)

1.[$\star\star\star\star$] Verifique que a propriedade `mul (xs++ys) = mul xs * mul ys` é satisfeita pelas funções `mul` e `(++)` para todas as listas finitas `xs` e `ys`.

```
mul [] = 1                    (mul1)
mul (x:xs) = x * mul xs       (mul2)

[] ++ ys = ys                 (++1)
(x:xs) ++ ys = x:(xs++ys)     (++2)
```

4. Grupo 4 (4.5 valores)

1.[⋆⋆] Usando a definição usual para uma árvore binária, define uma função `getLargestString` que recebe uma árvore de cadeias de caracteres, retorna a cadeia mais comprida contida na árvore. Se a árvore for vazia retorna uma *string* vazia.. Não pode usar funções de ordenação (`sort`...) para responder a esta pergunta.

2.[⋆] Defina um tipo `NaryTree` genérico:

1. cada nó tem um número arbitrário de filhos,
2. pode armazenar um tipo arbitrário em cada nó.

3.[⋆⋆] Um dicionário é uma estrutura de dados que permite aceder a definições de palavras. A cada palavra (`String`) corresponde uma definição (`String`). A palavra é designada por "chave" e a definição será o "valor" associado à chave. Defina um tipo `Entry` que associa uma palavra e a sua definição. Representa **uma** entrada do dicionário. O tipo deve ser instância das classes `Eq`, `Ord` e `Show`. A noção de igualdade e ordenação entre entradas é baseada na comparação das chaves. Poderá usar a definição por defeito da função `show`.

4.[⋆] Defina uma função que recebe um valor de tipo `Entry` (ver pergunta anterior) e escreve no ecrã a chave e o valor.

5.[⋆⋆⋆] Defina uma função `saveEntries` que recebe uma lista de valores de tipo `Entry` e pede ao utilizador, para cada valor se deve ou não ser gravada. Caso o utilizador responder com o carácter `s` a função grava a entrada correspondente no ficheiro `Dictionary.txt`.

Princípios de Programação

DI / FCUL, Exame de 2ª época

3 de fevreiro de 2015

Duração: 2h30
Cotação: 14 valores, $\star$ = 0.5 valores
Resolva cada grupo em páginas diferentes.
Justifique sempre as suas respostas.

1. Grupo 1 (2.5 valores)

1.[$\star$] Qual o tipo mais geral da expressão : `\x h -> h x ++ reverse x` ?

2.[$\star\star$] Qual deve ser a função `f` para que a avaliação da expressão

```
map ((/2) .  f) (zip [1 ..  4] [5 ..  13])
```

resulte em `[2.5,3.0,3.5,4.0]` ?

3.[$\star\star$] Como funciona a avaliação preguiçosa ? Exemplifique avaliando a expressão `take 3 (zip [0..] [0,-1..])` passo a passo. Quando é que seu uso é desvantajoso em termos de computação e/ou de memória ?

2. Grupo 2 (4.5 valores)

1.[$\star\star$]Defina a função `sumImp :: (Ord a, Int a) => [a] -> a`, que é a soma dos valores ímpares da lista dada. Deve implementar esta função usando a função `foldl`.

2.[$\star\star$] Considere uma matriz definida com uma lista de linhas (cada linha é representada por uma lista de valores numéricos). Defina uma função `minAndMax` que retorna um par com o valor máximo e o valor mínimo da matriz.

3.[$\star$] Defina a função `factors` que, dado um número inteiro retorna uma lista com todos os seus divisores. A sua função deve incluir uma lista de compreensão.

4.[$\star\star$] Defina a função `filter2::(a-> Bool) -> [a] -> [a]` que é o subconjunto da lista dada, onde fazem parte apenas os elementos que satisfaçam o predicado dado. A diferença em relação ao `filter` é que no `filter2` cada elemento deve aparecer em duplicado. Utilize o `foldr` ou o `foldl` para definir `filter2`. Por exemplo, `filter2 (>0) [-2..3]` é `[1,1,2,2,3,3]`.

5.[$\star\star$] Defina uma função `checkNum :: IO()` que lê do teclado uma sequência de caracteres e escreve no ecrã "`e um numero`" se a cadeia de caracteres for composta apenas por dígitos e "`nao e um numero`" caso contrário. Pode escrever funções auxiliares e usar a função `isDigit :: Char -> Bool`.

3. Grupo 3 (3 valores)

1.[⋆⋆⋆] Prove que `sum (xs ++ ys) = sum xs + sum ys` para qualquer listas `xs` e `ys`.

```
sum [] = 0                  -- (sum1)
sum (x:xs) = x + sum xs     -- (sum2)

(++) xs [] = xs                     -- (++1)
(++) [] ys = ys                     -- (++2)
(++) (x:xs) ys = x : (xs ++ ys)     -- (++3)
```

2.[⋆⋆⋆] Prove que `sum (reverse xs) = sum xs` para qualquer lista `xs`. Poderá usar o resultado da pergunta anterior.

```
reverse [] = []                         -- (rev1)
reverse (x:xs) = reverse xs ++ [x]      -- (rev2)
```

4. Grupo 4 (4 valores)

1.[⋆⋆] Defina o tipo `Shape` onde os valores possíveis podem ser círculos (definidos pelo seu raio), rectângulos (definidos pela altura e largura). O tipo `Float` deve ser usado para representar as medidas. O tipo deve instanciar as classes `Eq` e `Ord`. Dois valores de tipo `Shape` são iguais se são de mesmo tipo e dimensões. A relação de ordem é relativa a área das formas.

2.[⋆⋆⋆] Defina o tipo `ShapeTree` que representa uma árvore binária de pesquisa onde um valor de tipo `Shape` é armazenado em cada nó. Defina uma função `insertShape` que insere uma forma numa árvore.

3.[⋆⋆⋆] Escreve uma função `writeShapes` que, dado uma árvore de tipo `ShapeTree` e um nome de ficheiro, escreve o conteúdo da árvore num ficheiro com esse nome e onde cada forma está escrita numa linha. As linhas devem ser ordenadas por ordem decrescente (ver perguntas anteriores). Não use funções de ordenação na sua resposta.

Ciências ULisboa | Informática

Princípios de Programação
Exame Um
18 janeiro 2016

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* **`Prelude`**.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão; ordem superior. 3 valores]

a) Uma linha de texto pode ser representada por uma lista de palavras, onde cada palavra é uma **`String`**. Escreva, *utilizando recursão*, a definição da função `juntaLinha :: [String] -> String` que transforma uma linha numa **`String`** onde as palavras aparecem *separadas* por espaços. Por exemplo, `juntaLinha ["bom", "dia"] = "bom␣dia"`.

b) A função `apagaDuplicados` apaga os elementos duplicados *adjacentes* de uma dada lista. Por exemplo,
`apagaDuplicados [1,2,2,3,3,3,1,1] = [1,2,3,1]`. Defina a função `apagaDuplicados` utilizando **`foldl`** ou **`foldr`**.

c) A função **`filter`** pode ser definida em termos de **`concat`** e **`map`**:

```
filter p = concat . map caixa
  where caixa x = ...
```

Escreva uma definição para a função `caixa`.

Grupo 2. [Tipos de dados abstratos. 3 valores]

Um árvore binária pode ser implementada por um tipo algébrico da seguinte forma:

```
data Arvore a = Folha | No a (Arvore a) (Arvore a)
```

a) É fácil de ver que árvores diferentes podem conter os mesmos elementos. Apresente dois termos Haskell diferentes, ambos do tipo `Arvore Int`, mas que contenham os mesmos elementos (isto é os mesmos números inteiros).

b) Torne o tipo de dados `Arvore` um caso particular (**`instance`**) da classe de tipos **`Eq`** sabendo que se pretende que duas árvores sejam iguais sempre que contiverem os mesmos elementos.

c) Uma árvore de pesquisa é um objecto do tipo `Arvore a` que obedece às seguintes propriedades:

- Uma árvore `Folha` é de pesquisa.
- Uma árvore `(No raiz esq dir)` está ordenada se a) todos os valores em `esq` forem menores que `raiz` e b) todos os valores em `dir` forem maiores que `raiz` e c) as árvores `esq` e `dir` forem de pesquisa.

Escreva uma função `minimo` que, dada uma árvore de pesquisa de elementos ordenáveis, devolve o mais pequeno elemento na árvore. Mais precisamente, se árvore for vazia devolve **`Nothing`**, caso contrário devolve **`Just`** `v`, onde `v` é o menor elemento na árvore. Escreva uma função $\mathcal{O}(\log n)$, onde n é o número de elementos na árvore. Não se esqueça de escrever a assinatura da função.

Grupo 3. [Teste. 3 valores]

Suponha dada uma função `elementos :: Arvore a -> [a]` que transforma uma árvore (tal como definida no Grupo **2**) numa lista, utilizando o *percurso infixo*. Relembre que a visita infixa de uma árvore percorre primeiro a sub-árvore esquerda, depois a raiz da árvore e finalmente a sub-árvore direita. Assuma também que o tipo de dados `Arvore` é um caso particular (**`instance`**) da classe de tipos `Arbitrary`.

a) Considere a seguinte propriedade: *O mais pequeno elemento de uma árvore pertence aos elementos da árvore*. Escreva uma propriedade QuickCheck que relacione as funções `elementos` e `minimo` (tal como definida no Grupo **2**).

b) Considere a agora a propriedade: *A lista obtida pelo percurso infixo de uma árvore de pesquisa está ordenada*. Escreva uma propriedade QuickCheck que verifica esta propriedade. Note que a ação `arbitrary` para tipo `Arvore` devolve uma árvore que não é necessariamente de pesquisa. Suponha dada a função `ehDePesquisa :: Arvore a -> Bool` que verifica se uma dada árvore é de pesquisa.

c) Torne o tipo de dados `Arvore` um caso particular (**`instance`**) da classe `Arbitrary`.

Grupo 4. [Raciocínio sobre programas. 2 valores]

Mostre que

```
map f (xs ++ ys) = map f xs ++ map f ys
```

para todas as listas finitas `xs` e `ys` e todas as funções `f`. Eis as definições das duas funções envolvidas:

```
map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x : map f xs

(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys
(x:xs) ++ ys = x : xs ++ ys
```

Grupo 5. [Expressões lambda em Java. 3 valores]

a) Escreva um método que emula o funcionamento da função Haskell **`foldl`**, com a seguinte assinatura:

```
static <T, R> R foldl(
  BiFunction<R, T, R> funcao, R zero, List<T> lista)
```

Por exemplo, o seguinte pedaço de código

```
List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
System.out.println (
  foldl((acc, e) -> acc + e * e, 0, numeros));
```

deverá imprimir `140`.

b) Utilizando a função `foldl` criada na alínea anterior, escreva agora um método que receba uma função de comparação e uma lista, e que calcule o elemento máximo desta, tal como determinado pela função de comparação. Eis a assinatura do método pretendido:

```
static <T> T maximo(Comparator<T> comparador, List<T> lista)
```

Por exemplo, o seguinte pedaço de código

```
List<String> idades =
  Arrays.asList("Ana,30", "Rui,60", "Luis,45", "Maria,55");
Comparator<String> maisVelho = Comparator.comparing(
  s -> Integer.valueOf(s.split(",")[1]));
System.out.println(maximo(maisVelho, idades));
```

deverá imprimir `Rui,60`.

Ciências ULisboa | Informática

Princípios de Programação
Exame Dois
2 fevereiro 2016

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* **`Prelude`**.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão; ordem superior. 3 valores]

Considere a função **`zip3`** que junta três listas, do mesmo modo que a função **`zip`** junta duas. Por exemplo:

```
ghci> zip3 [1,2,3] "abcd" [True,False]
[(1,'a',True),(2,'b',False)]
```

a) Escreva uma definição para a função utilizando recursão.

b) Escreva agora uma definição da mesma função recorrendo às funções **`zip`** e/ou **`zipWith`**.

c) Defina a função **`unzip`** `:: [(a, b)] -> ([a], [b])` utilizando a função **`foldl`** ou **`foldr`**.

Grupo 2. [Tipos de dados abstratos. 3 valores]

Considere o tipo de expressões aritméticas simples, construidas a partir de números inteiros, utilizando os operadores de adição e multiplicação.

```
data Exp = Val Int | Soma Exp Exp | Produto Exp Exp
```

a) Escreva uma função `valor` que calcula o valor de uma expressão aritmética. Exemplo para calcular o valor de $2 * 3 + 4$:

```
ghci> valor (Soma (Produto (Val 2) (Val 3)) (Val 4))
10
```

b) Torne o tipo de dados `Exp` instância da classe **`Show`**, de modo a que a representação textual de uma `Exp` contenha os operadores `+` e `*` em modo infixo, como é comum apresentar na aritmética. Para simplificar o exercício, apresentamos as expressões com todos os parêntesis. Por exemplo:

```
ghci> Soma (Produto (Val 2) (Val 3)) (Val 4)
"((2*3)+4)"
```

c) Defina uma função `estatistica` que calcule o número de operadores de multiplicação, o número de operadores de soma e o número de valores inteiros que ocorrem numa dada expressão aritmética. Por exemplo,

```
ghci> estatistica (Soma (Produto (Val 2) (Val 3)) (Val 4))
(1,1,3)
```

Grupo 3. [Entrada e saída. 3 valores]

a) Escreva uma função `lerInteiros :: String -> IO [Integer]` que, dado o nome de um ficheiro, devolva uma lista de números inteiros lidos do ficheiro. Assuma que o ficheiro contém um número inteiro por linha e que contém pelo menos uma ocorrência do número zero. A função deverá devolver a lista dos números inteiros constantes no ficheiro até encontrar o número zero (o zero não deverá fazer parte da lista).

Por exemplo, se o ficheiro `numeros.txt` contiver os inteiros `1 2 3 4 0 5 6` (um por linha), a expressão `lerInteiros "numeros.txt"` deverá devolver a lista `[1,2,3,4]`.

b) Utilizando a função da alínea anterior, construa agora um programa `main :: IO ()` que leia o nome de um ficheiro da linha de comandos e que imprima a soma dos inteiros que aparecem no ficheiro até à primeira ocorrência do número zero.

Grupo 4. [Raciocínio sobre programas. 2 valores]

Mostre a seguinte lei para todas as listas finitas *não vazias* `xs` e todas as funções `f`.

```
last (map f xs) = f (last xs)
```

Eis as definições das duas funções envolvidas:

```
last :: [a] -> a
last [x]    = x
last (_:xs) = last xs

map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x : map f xs
```

Sugestão: Para o caso base utilize uma lista com um elemento apenas.

Grupo 5. [Expressões lambda em Java. 3 valores]

a) Implemente em Java a função `filter` com a seguinte assinatura

```
static <T> List<T> filter(Predicate<T> predicado,
                          List<T> lista)
```

utilizando para tal a seguinte função que se considere dada:

```
static <T, R> R foldl(BiFunction<R, T, R> funcao,
                      R zero, List<T> lista)
```

O comportamento da função `filter` deverá ser idêntico ao da função Haskell com o mesmo nome.

b) Implemente em Java a função `compose` à semelhança do operador de composição `.` em Haskell, com a seguinte assinatura

```
static <T1, T2, T3> Function<T1, T3>
  compose(Function<T2, T3> f, Function<T1, T2> g)
```

c) Utilizando a função `compose` da alínea anterior, crie uma nova função que, dada uma lista de inteiros, calcule uma nova lista apenas com os números inferiores a 50 existentes na lista original, incrementados de uma unidade. A sua função deve utilizar as funções `map` e `filter`. Por exemplo, dada a seguinte lista

```
List<Integer> numeros =
  Arrays.asList(65, 12, 33, 85, 44, 50, 70, 45, 59, 23);
```

a nova função deverá retornar uma lista contendo os números `13, 34, 45, 46, 24`.

Considere dada a função `map` com a seguinte assinatura:

```
static <T, R> List<R> map(Function<T, R> funcao,
                          List<T> lista)
```

Ciências ULisboa | Informática

PRINCÍPIOS DE PROGRAMAÇÃO

Exame1 16/Janeiro/2017
Cotação 14 valores
Duração 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*2 valores*] Para cada alínea abaixo responda e justifique:

   a) Qual o valor da expressão `fst (tail "9998", 99)`?

   b) Qual o tipo mais geral da expressão `\(x,y,z) -> x == z * snd y`?

   c) Defina, em Haskell, um tipo que represente polinómios com coeficientes inteiros. Como representa $2x^4 + x - 1$ no tipo que definiu?

   d) O que é um tipo recursivo? Mostre um exemplo.

2) [*6 valores*] Defina as seguintes funções (pode utilizar as funções do `Prelude`):

   a) `countExists :: (a -> b -> Bool) -> a -> [b] -> Int`, o número de ocorrências em que o predicado binário é `True` quando aplicado ao elemento dado como primeiro argumento juntamente com cada um dos elementos da lista dada. Por exemplo,

```
Main> countExists (\x y -> x + y == 102) 100 [1,2,0,2,2]
3
```

   b) `writeStr :: String -> IO()`, que mostra a frase dada no ecrã. Nesta alínea não pode utilizar `putStr` e `putStrLn` mas pode usar `putChar::Char -> IO()`.

   c) Tendo o seguinte tipo de árvore binária:

```
data Tree a = Null | Node (Tree a) a (Tree a)
```

   defina o predicado `findElem :: Ord a => Tree a -> a -> Bool`, que verifica se o elemento dado encontra-se na árvore dada. Esta árvore binária é de pesquisa (ie, a sub-árvore esquerda de cada nó contém elementos menores que o nó, e a sub-árvore direita contém os elementos maiores).

3) [*3.5 valores*] Seja a função `collapse` que, dada uma função binária `f` e uma lista, devolve uma lista com os resultados de aplicar `f` a cada par consecutivo de elementos da lista original. Se a lista original conter um número ímpar de elementos, o último elemento não é utilizado.

   Por exemplo, a aplicação `collapse (*) [1..9]` devolve `[2,12,30,56]`, ou seja o resultado de `[1*2,3*4,5*6,7*8]`, tendo sido descartado o último elemento.

   a) Escreva a especificação da função `collapse` usando `length` e `(!!)`.

   b) Implemente a função de acordo com a especificação

   c) Escreva testes QuickCheck que testem a especificação dada em a). Nos testes utilize a função `(+)` como exemplo da função binária `f`.

   d) Se avaliarmos os testes sem encontrar contra-exemplos, o que podemos afirmar?

4) [*2.5 valores*] Defina o tipo `Point2D` que descreve um ponto no plano cartesiano de coordenadas inteiras. Torne-o uma instância das classes `Eq`, `Ord` e `Arbitrary`. Dois pontos são iguais se tiverem as mesmas coordenadas. Considere que um ponto P1 é menor que outro P2 se P1 estiver mais perto da origem que P2 (considere como distância a soma dos quadrados das coordenadas). Para o gerador aleatório crie pontos com coordenadas entre `-100` e `100`.

Ciências ULisboa | Informática

PRINCÍPIOS DE PROGRAMAÇÃO

Exame2 31/Janeiro/2017
Cotação 14 valores
Duração 2h30m

Escreva o seu nome e número em todas as folhas. Numere as folhas e indique o número total de folhas entregues na folha de rosto. Leia o exame com atenção e justifique sempre as suas respostas.

1) [*2 valores*] Para cada alínea abaixo responda e justifique:

   a) Qual o valor da expressão `map ((*2).fst) [(1,2),(6,1),(3,3)]`?

   b) Qual o tipo mais geral da expressão `\xs x -> foldl (*) x (xs++[1])`?

   c) Defina uma função com tipo `(Eq a, Num a) => (a -> Bool) -> a -> a`.

   d) O que significa uma função ter recursão terminal (também designado por recursão na cauda)? Mostre um exemplo.

2) [*6 valores*] Defina as seguintes funções:

   a) `digitos :: Int -> [Int]` que dado um inteiro positivo devolve a lista dos seus dígitos por ordem inversa. Por exemplo, `digitos 1243` deve devolver a lista `[3,4,2,1]`.

   b) Dizemos que um inteiro positivo é especial se for igual à soma dos cubos dos seus dígitos. Por exemplo, 153 é especial porque $1^3+5^3+3^3 = 153$. Defina a função `especial :: Int -> Bool` que verifica se um dado inteiro positivo é especial.

   c) Defina o programa `pedirEspeciais :: IO()` que peça inteiros positivos ao utilizador e informe se são números especiais (escrevendo no ecrã `SIM` ou `NAO`). O programa deve terminar quando o utilizador introduzir o valor zero (não há necessidade de definir um programa de IO robusto).

3) [*3 valores*] Sejam as seguintes definições

```
sum []     = 0                    {sum1}
sum (x:xs) = x + sum xs           {sum2}

double []     = []                {dbl1}
double (x:xs) = 2*x : double xs   {dbl2}
```

   a) Prove que `sum (double xs) == 2 * sum xs`.

   b) Escreva uma propriedade QuickCheck que teste o predicado a).

   c) Escreva outra propriedade QuickCheck que teste que, após aplicar `double` a uma lista não vazia, o máximo elemento da lista resultante é o dobro do máximo da lista original.

4) [*3 valores*] Defina o tipo `Data` que descreve um dia do ano (ie, dia, mês e ano). O valor do mês deve ser representado por um enumerado. Torne o tipo `Data` uma instância das classes `Eq`, `Show` e `Arbitrary`.

   As datas devem ser apresentadas com o mesmo formato deste exemplo: "`31 de Janeiro de 1990`".

   Para o gerador aleatório assuma válidas apenas datas entre 1970 e 2016. Tem de ter atenção que nem todos os meses têm os mesmos dias. Assuma também que tem acesso à função `bissexto :: Int -> Bool`.

Ciências ULisboa | Informática

Princípios de Programação
Exame Um
5 janeiro 2018

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, os sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* `Prelude`.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão; ordem superior. 3 valores]

Note bem: Para este grupo só pode utilizar funções constantes no `Prelude`.

a) Escreva uma função

```
encontra :: (a -> Bool) -> [a] -> Maybe a
```

que receba um predicado e uma lista e que devolva o primeiro elemento para o qual o predicado é verdadeiro. A função devolve `Nothing` se não houver nenhum elemento nestas condições. Exemplos:

```
ghci> encontra isUpper "a1 A#C"
Just 'A'
ghci> encontra (<0) [0..23]
Nothing
```

b) Escreva uma função

```
fatias :: Int -> [a] -> [[a]]
```

que divida uma lista em componentes de comprimento k. A última fatia pode ser mais curta do que as outras fatias, dependendo do comprimento da lista de entrada. Exemplos:

```
>>> fatias 3 "olacomovais"
["ola","com","ova","is"]
>>> fatias 4 "haskell.org"
["hask","ell.","org"]
```

c) Escreva uma função

```
separa :: Eq a => [a] -> [a] -> [[a]]
```

que, dada uma lista de elementos separadores e uma lista de elementos a separar, devolva uma lista com todas as sequências delimitadas por um ou mais separadores. Exemplo:

```
ghci> separa " \t\n" "As armas\te\tos barões\n   que"
["As","armas","e","os","barões","que"]
```

Grupo 2. [Tipos de dados abstratos. 3 valores]

`Doc` é um tipo de dados algébrico que representa documentos de texto bem formados.

```
data Doc = Vazio
         | Texto String
         | NovaLinha
         | Concat Doc Doc
```

a) Torne o tipo de dados `Doc` instância da classe de tipos **`Show`**, de tal modo que: a `Vazio` corresponda a **`String`** vazia, que a `NovaLinha` correspomda uma mudança de linha (`"\n"`), e que `Concat` junte dois pedaços de texto num só. Por exemplo:

```
ghci> show $ Concat (Concat (Texto "As ") (Texto "armas"))
        (Concat Vazio NovaLinha)
"As armas\n"
```

b) Torne agora o tipo de dados `Doc` instância da classe de tipos **`Eq`**, de modo a que dois documentos sejam iguais se tiverem a mesma representação textual. Por exemplo:

```
ghci> Concat (Concat (Texto "As ") (Texto "armas"))
        (Concat Vazio NovaLinha) ==
          Concat Vazio (Texto "As armas\n")
True
```

Grupo 3. [Teste. 3 valores]

a) Torne o tipo de dados `Doc` do exercício anterior instância da classe de tipos `Arbitrary`.

b) Escreva propriedades `QuickCheck` que verifiquem as seguintes condições:

- A concatenação de qualquer documento com o documento vazio (`Vazio`) é igual ao documento original, e vice-versa, a concatenação de qualquer documento vazio com um dado documento é igual este documento.

- A operação de concatenação de documentos é associativa.

Grupo 4. [Raciocínio sobre programas. 3 valores]

Mostre a propriedade distributiva da multiplicação em relação à adição generalizada. Mais precisamente, mostre que a seguinte lei

```
n * sum xs = sum (map (n*) xs)
```

é verdadeira para todo o número `n` e para a toda a lista finita de números `xs`. Assuma válida a propriedade distributiva da multiplicação em relação à adição comum, $n(x + y) = nx + ny$.

Relembre a definição das seguintes funções constantes no **`Prelude`**, quando aplicadas a listas de números inteiros.

```
sum :: [Int] -> Int
sum []     = 0                  // sum1
sum (x:xs) = x + sum xs         // sum2

map :: (a -> b) -> [a] -> [b]
map _ []     = []               // map1
map f (x:xs) = f x : map f xs   // map2
```

Grupo 5. [Programação funcional em Java. 2 valores]

Escreva uma expressão Java que calcule a soma dos quadrados ímpares dos números compreendidos entre 1 e um dado número `n`, inclusivé. Por exemplo, o código abaixo deverá imprimir `35` se as reticências (`...`) denotarem a expressão pretendida neste exercício.

```
int n = 6
int exp = ...
System.out.print(exp)
```

Ciências ULisboa | Informática

Princípios de Programação
Exame Dois
24 janeiro 2018

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, os sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* `Prelude`.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão. 3 valores]

A cifra Bifid foi inventada por Felix Delastelle por volta de 1901. Eis como funciona. Começa-se por desenhar uma tabela da seguinte forma, onde o I e o J partilham a mesma posição.

	1	2	3	4	5
1	B	G	W	K	Z
2	Q	P	N	D	S
3	I/J	O	A	X	E
4	F	C	L	U	M
5	T	H	Y	V	R

A mensagem é convertida nas suas coordenadas (primeiro a linha e depois a coluna), mas estas são escritas verticalmente:

```
H A S K E L L
5 3 2 1 3 4 4
2 3 5 4 5 3 3
```

Depois as coordenadas são lidas em linhas:

```
5 3 2 1 3 4 4 2 3 5 4 5 3 3
```

Em seguida divide-se a lista de números em pares. Finalmente os pares são transformados em letras novamente, utilizando o quadrado. Obtém-se assim a palavra cifrada.

```
53 21 34 42 35 45 33
Y  Q  X  C  E  M  A
```

Assuma que são dadas as seguintes funções para aceder às linhas e colunas de uma tabela Bifid.

```
linhaBifid :: Char -> Int
colunaBifid :: Char -> Int
```

```
ghci> linhaBifid 'X'
3
ghci> colunaBifid 'X'
4
```

a) Escreva uma função, que dada uma palavra devolva um par de listas com as coordenadas da palavra. Assuma que a *string* de entrada contém apenas letras maiúsculas.

```
ghci> coordenadas "HASKELL"
([5,3,2,1,3,4,4],[2,3,5,4,5,3,3])
```

b) Escreva uma função que dada uma lista de inteiros, devolva a *string* correspondente. Assuma que a lista de entrada contém um número par de números entre 1 e 5.

```
ghci> palavra [5,3,2,1,3,4,4,2,3,5,4,5,3,3]
"YQXCEMA"
```

c) Utilizando as funções da alínea anterior escreva função `bifid`.

```
ghci> bifid "HASKELL"
"YQXCEMA"
```

Grupo 2. [Tipos de dados abstratos. 3 valores]

As expressões aritméticas podem ser escritas como expressões Haskell do tipo `Exp`.

```
data Exp = Variavel Char | Inteiro Int | Mais Exp Exp
   | Vezes Exp Exp
```

Por exemplo, a expressão $3(x + 2y)$ pode ser escrita como a expressão Haskell seguinte.

```
let expr = Vezes (Inteiro 3) (Mais (Variavel 'x') (
   Vezes (Inteiro 2) (Variavel 'y')))
```

a) O comprimento de uma expressão é o número de operadores binários (`Mais` e `Vezes`) contidos na expressão. Escreva a função `comprimento`. Por exemplo:

```
ghci> comprimento expr
3
```

b) Escreva uma função que avalie uma expressão aritmética. Os valores das variáveis são dados por uma *lista de associação*. Uma lista de associação é uma lista de pares, onde o primeiro elemento funciona como chave (o nome da variável) e o segundo como o valor associado à chave (o valor da variável). Tome zero para o valor de cada variável ausente na lista. Por exemplo:

```
ghci> avalia [('y',2)] expr
12
```

Sugestão: utilize a função **`lookup`** constante no **`Prelude`** que procura o valor de uma chave numa lista associação.

```
lookup :: Eq a => a -> [(a, b)] -> Maybe b
```

c) Torne o tipo `Exp` instancia da classe de tipos **`Show`**, onde a) omitimos o operador de multiplicação (como é usual em matemática) e b) colocamos parêntesis *apenas* à volta do produto de uma soma.

Por exemplo:

```
ghci> Vezes (Vezes (Variavel 'x') (Inteiro 5)) (Variavel 'x')
x5x
ghci> expr
3(x+2y)
```

Grupo 3. [Teste. 3 valores]

a) Torne o tipo `Exp` definido no grupo anterior instância da classe de tipos `Arbitrary`.

b) Escreva em QuickCheck uma propriedade que assegure que o comprimento de uma expressão é um número não negativo.

c) Escreva em QuickCheck uma propriedade para verificar que o produto de dois números positivos é positivo.

Grupo 4. [Entrada e saída. 2 valores]

Escreva uma função `calculadora` que repetidamente lê expressões aritméticas (uma por linha) até encontrar uma expressão que avalia para zero. Neste ponto o programa deve terminar. Para cada expressão lida o programa deve imprimir o valor da expressão. Assuma que o tipo de dados `Exp` definido acima é instância da classe de tipos **`Read`**. Os valores das variáveis são dados por uma lista associação (ver grupo 2, alínea b). Por exemplo:

```
ghci> calculadora [('y',2)]
3(x + 2y)
12
3z
0
ghci>
```

Grupo 5. [Raciocínio sobre programas. 3 valores]

Mostre que a inversa da inversa de uma lista é a lista original. Mais precisamente mostre que a lei

```
reverse (reverse xs) = xs
```

é verdadeira para a toda a lista finita `xs`. Assuma válida a seguinte propriedade

```
reverse (ys ++ [x]) = x : reverse ys -- prop1
```

para a toda a lista finita `ys` e elemento `x`.

Relembre a definição da função **`reverse`** constante no **`Prelude`**.

```
reverse :: [a] -> [a]
reverse []     = []                          -- rev1
reverse (x:xs) = reverse xs ++ [x]           -- rev2
```

Ciências ULisboa | Informática

Princípios de Programação
Exame Um
11 janeiro 2019

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, os sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* **`Prelude`**.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão. 3 valores]

Note bem: Para este grupo só pode utilizar funções constantes no **`Prelude`**.

a) Escreva uma função `multiplosEntre` que receba três inteiros `a`, `b` e `c`, e que devolva o número de valores que estejam entre `a` e `b` e que sejam múltiplos de `c`. Admita que `a` é sempre menor ou igual a `b`. Exemplos:

```
ghci> multiplosEntre 0 1000 2
500
ghci> multiplosEntre 0 12 3
4
```

b) Escreva uma função `intercalar :: [a] -> [[a]] -> [a]` que receba uma lista `xs` e uma lista de listas `yss` e que concatene os elementos de `yss` colocando `xs` entre cada par de elementos de `yss`. Por exemplo:

```
ghci> intercalar "_" ["As", "armas", "e", "os", "barões"]
"As_armas_e_os_barões"
```

c) Escreva uma função `transposta :: [[a]] -> [[a]]` que transponha as linhas e as colunas do seu argumento. Exemplos:

```
ghci> transposta ["azul", "verde", "roxo"]
["avr","zeo","urx","ldo","e"]
ghci> transposta ["verde", "encarnado"]
["ve", "en", "rc", "da", "er", "n", "a", "d", "o"]
```

Grupo 2. [Tipos de dados abstratos. 3 valores]

Considere o seguinte tipo de dados que representa árvores DOM de um *browser*:

```
data Html = Div [Html] | Texto String | Negrito String
```

a) Escreva uma função `profundidade :: Html -> Int` que devolva a profundidade de uma árvore `Html`. A profundidade de `Texto`, de `Negrito` e de `Div` sem sub-árvores é 1. Caso contrário, a profundidade da árvore é 1 mais o máximo das profundidades das suas sub-árvores. Exemplos:

```
ghci> profundidade $ Negrito "olá"
1
ghci> profundidade $ Div []
1
ghci> profundidade $ Div [Div [Texto "olá"], Negrito "adeus"]
3
```

b) Escreva uma função `realcar :: String -> Html -> Html` que, dada uma *string* `s` e uma árvore `Html a`, devolva uma cópia de `a` substituindo todas as ocorrências de `Texto s` por `Negrito s`. Exemplo:

```
ghci> realcar "olá" $ Div [Texto "olá", Texto "adeus"]
Div [Negrito "olá", Texto "adeus"]
```

c) Torne o tipo `Html` uma instância da classe de tipos `Show`, de forma a que a sua representação textual seja da forma mostrada abaixo.

```
ghci> Div [Texto "Este e o ",
  Negrito "meu", Texto " primeiro blog post!"]
"<div>Este e o <b>meu</b> primeiro blog post!</div>"
```

Grupo 3. [Teste. 3 valores]

a) Torne o tipo de dados `Html` definido no grupo 2 uma instância da classe de tipos `Arbitrary`.

b) Escreva a seguinte propriedade quickCheck:

> A profundidade de uma árvore `Html` é sempre maior do que a profundidade de qualquer das suas sub-árvores, caso as tenha.

c) Escreva a seguinte propriedade quickCheck:

O comprimento da *string* representação textual (obtida com **show**) de uma árvore `Html` é sempre maior ou igual à soma dos comprimentos das representações textuais das suas sub-árvores.

Grupo 4. [Raciocínio sobre programas. 3 valores]

Considere a função `numDe` que calcula o número de ocorrências de um dado elemento numa lista.

```
numDe :: a -> [a] -> Int
numDe _ []  = 0                                  -- numDe/1
numDe x (y:ys)
  x == y    = 1 + numDe x ys                     -- numDe/2
  otherwise = numDe x ys                         -- numDe/3
```

Relembre também a definição da função **length** constante no **Prelude**:

```
length :: [a] -> Int
length []     = 0                              -- length/1
length (_:xs) = 1 + length xs                  -- length/2
```

Mostre que

```
numDe x ys <= length ys
```

para todos os elementos `x` e todas as listas finitas `ys`.

Grupo 5. [Programação funcional em Java. 2 valores]

Preencha a expressão Java em falta (assinalada com ⊡) de modo a que o código abaixo imprima o valor máximo que a função `f` toma no intervalo [`inicio`, `fim`]. A função deve ser calculada nos pontos entre `inicio` e `fim` com incrementos de `inc`.

```
Function<Double, Double> f = x -> - x * x + 2;
double inicio = -2;
double fim = 2;
double inc = 0.0001;
double maxF = ⊡;
System.out.println(maxF);
```

Para os valores de `f`, `inicio`, `fim` e `inc` dados, a execução do código deverá imprimir:

```
2.0
```

Ciências ULisboa | Informática

Princípios de Programação
Exame Dois
30 janeiro 2019

Material junto de si: *Deverá ter consigo* apenas *canetas e o cartão de estudante. Os casacos, os sacos e as mochilas com o restante material, incluindo telemóveis, deverão ser deixados junto ao quadro. A violação destas regras implica a anulação do exame.*
Sobre a escrita do exame: *Inicie cada novo grupo numa* página *separada. Junte assinaturas para todas as funções que escrever. Pode utilizar qualquer função da biblioteca Haskell. Não se esqueça de importar o módulo quando a função não estiver no* **`Prelude`**.
Duração: *Duas horas e trinta minutos.*

Grupo 1. [Recursão; ordem superior. 3 valores]

a) Escreva uma função que calcule o máximo de uma função inteira num dado intervalo. O intervalo é dado por um par de inteiros. Assuma que o intervalo contém pelo menos um ponto. Utilize a seguinte assinatura:
`maximo:: (Int -> Int)-> (Int, Int)-> Int`. Exemplos:

```
ghci> maximo (\x -> 2*x) (0,10)
20
ghci> maximo (\x -> -x) (0,10)
0
```

b) Escreva a função **`map`** `:: (a -> b)-> [a] -> [b]` utilizando uma das variantes da função `fold`.

c) Escreva uma função `verificaContribuinte` que verifique se um número inteiro representa um contribuinte válido. A função recebe um número inteiro com exactamente 9 algarismos. A verificação deverá ser feita do seguinte modo: seja s a soma dos produtos do 8º algarismo por 2, do 7º algarismo por 3, do 6º algarismo por 4, do 5º algarismo por 5, do 4º algarismo por 6, do 3º algarismo por 7, do 2º algarismo por 8 e do 1º algarismo por 9. Seja r o resto da divisão inteira de s por 11. Se r for 0 ou 1, o último algarismo terá de ser 0. Caso contrário, o último algarismo deverá ser 11 subtraído de r. Exemplos:

```
ghci> verificaContribuinte 502618418
True
ghci> verificaContribuinte 502618411
False
```

Grupo 2. [Tipos de dados abstratos. 3 valores]

Considere a seguinte estrutura de dados algébrica que representa uma série de coloridas bonecas Matriosca, feitas geralmente de madeira e colocadas umas dentro das outras. No nosso caso, cada boneca tem uma cor dominante, cor esta que está representada no tipo de dados `Matriosca`.

```
data
Cor = Branco | Azul | Vermelho | Verde
  deriving Show
data
 Matriosca = M Cor (Maybe Matriosca)
  deriving Show
```

a) Escreva uma função que transforme uma série de bonecas Matriosca numa lista de cores. O primeiro elemento da lista deve ser a cor da boneca mais exterior. Por exemplo:

```
ghci> paraLista $ M Branco (Just (M Azul (Just (M
  Vermelho (Just (M Verde Nothing))))))
[Branco, Azul, Vermelho, Verde]
```

b) Equipe o tipo de dados `Cor` com a menor relação de equivalência **`Eq`** gerada pelo par `Azul == Vermelho`. Relembre que uma relação de equivalência deve ser reflexiva ($a = a$), simétrica (se $a = b$, então $b = a$) e transitiva (se $a = b$ e $b = c$, então $a = c$), pelo que deve incluir na relação outros pares para além do dado.

c) Escreva uma função que devolva o número bonecas com uma dada cor contida numa série de bonecas Matriosca. Por examplo:

```
ghci> numCor Azul $ M Branco (Just (M Azul (Just
  (M Vermelho (Just (M Azul Nothing))))))
2
```

Grupo 3. [Raciocínio sobre programas. 3 valores]

Considere as seguintes funções sobre Matrioscas e sobre listas.

```
numMatrioscas :: Matriosca -> Int
numMatrioscas (M c Nothing)  = 1
numMatrioscas (M c (Just m)) = 1 + (numMatrioscas m)

paraMatriosca :: [Cor] -> Matriosca
paraMatriosca [x]    = M x Nothing
paraMatriosca (x:xs) = M x (Just $ paraMatriosca xs)
```

```
length :: [a] -> Int
length []     = 0
length (_:xs) = 1 + length xs
```

Mostre que

```
numMatrioscas (paraMatriosca xs)= length xs
```

para todas as listas finitas e **não vazias** `xs`.

Grupo 4. [Entrada e saída. 3 valores]

Escreva um programa executável que leia do *standard input* várias cores de forma a construir uma série de Matrioscas, da mais interior para a mais exterior. Assim que uma linha não contenha uma cor válida (`"Verde"`, `"Azul"`, `"Vermelho"` ou `"Branco"`), o programa deverá terminar. Antes de o fazer, porém, deverá imprimir a Matriosca construída. Eis um exemplo de execução do programa `./criarMatriosca`. Se inserirmos na consola o seguinte *input*:

```
Verde
Azul
Branco
xpto
```

obtemos o seguinte *output*:

```
M Branco (Just (M Azul (Just (M Verde Nothing))))
```

Grupo 5. [Programação funcional em Java. 2 valores]

Escreva um método que receba um *stream* de tipo genérico, um predicado de elementos do mesmo tipo e ainda uma função também de elementos do mesmo tipo. O método deve devolver um novo *stream* que contenha cada elemento do *stream* original convertido pela função caso passe o predicado. Os elementos que não passem o predicado devem aparecer inalterados no *stream* saída. Por exemplo, o seguinte código

```
converter(Stream.of(-3, 3, -2, 2),
    (Integer x) -> x > 0,
    (Integer x) -> x * x)
  .forEach(System.out::println);
```

deverá produzir o seguinte *output*

```
-3
9
-2
4
```