

# Cálculo de Programas Trabalho Prático MiEI+LCC — Ano Lectivo de 2016/17

*Departamento de Informática*  
Universidade do Minho

Junho de 2017

<b>Grupo nr.</b>	<b>13</b>
a77486	Jose Pedro Resende
a79007	Patrícia Barreira
a78225	Miguel Leite

## Conteúdo

<b>1</b>	<b>Preâmbulo</b>	<b>2</b>
<b>2</b>	<b>Documentação</b>	<b>2</b>
<b>3</b>	<b>Como realizar o trabalho</b>	<b>3</b>
<b>A</b>	<b>Mónade para probabilidades e estatística</b>	<b>10</b>
<b>B</b>	<b>Definições auxiliares</b>	<b>11</b>
<b>C</b>	<b>Soluções propostas</b>	<b>11</b>

# 1 Preâmbulo

A disciplina de Cálculo de Programas tem como objectivo principal ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de um repertório de *combinadores* que formam uma álgebra da programação (conjunto de leis universais e seus corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos dois cursos que têm esta disciplina, restringe-se a aplicação deste método ao desenvolvimento de programas funcionais na linguagem **Haskell**.

O presente trabalho tem por objectivo concretizar na prática os objectivos da disciplina, colocando os alunos perante problemas de programação que deverão ser abordados composicionalmente e implementados em **Haskell**. Há ainda um outro objectivo: o de ensinar a documentar programas e a produzir textos técnico-científicos de qualidade.

## 2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita “*literária*” [3], cujo princípio base é o seguinte:

*Um programa e a sua documentação devem coincidir.*

Por outras palavras, o código fonte e a sua documentação deverão constar do mesmo documento (ficheiro).

O ficheiro `cp1617t.pdf` que está a ler é já um exemplo de *programação literária*: foi gerado a partir do texto fonte `cp1617t.lhs`<sup>1</sup> que encontrará no *material pedagógico* desta disciplina descompactando o ficheiro `cp1617t.zip` e executando

```
lhs2TeX cp1617t.lhs > cp1617t.tex
pdflatex cp1617t
```

em que `lhs2tex` é um pre-processor que faz “pretty printing” de código Haskell em **L<sup>A</sup>T<sub>E</sub>X** e que deve desde já instalar a partir do endereço

<https://hackage.haskell.org/package/lhs2tex>.

Por outro lado, o mesmo ficheiro `cp1617t.lhs` é executável e contém o “kit” básico, escrito em **Haskell**, para realizar o trabalho. Basta executar

```
ghci cp1617t.lhs
```

para ver que assim é:

```
GHCI, version 8.0.2: http://www.haskell.org/ghc/  :? for help
[ 1 of 11] Compiling Show           ( Show.hs, interpreted )
[ 2 of 11] Compiling ListUtils      ( ListUtils.hs, interpreted )
[ 3 of 11] Compiling Probability   ( Probability.hs, interpreted )
[ 4 of 11] Compiling Cp             ( Cp.hs, interpreted )
[ 5 of 11] Compiling Nat             ( Nat.hs, interpreted )
[ 6 of 11] Compiling List             ( List.hs, interpreted )
[ 7 of 11] Compiling LTree          ( LTree.hs, interpreted )
[ 8 of 11] Compiling St              ( St.hs, interpreted )
[ 9 of 11] Compiling BTree          ( BTree.hs, interpreted )
[10 of 11] Compiling Exp             ( Exp.hs, interpreted )
[11 of 11] Compiling Main              ( cp1617t.lhs, interpreted )
Ok, modules loaded: BTree, Cp, Exp, LTree, List, ListUtils, Main, Nat,
Probability, Show, St.
```

O facto de o interpretador carregar as bibliotecas do *material pedagógico* da disciplina, entre outras, deve-se ao facto de, neste mesmo sítio do texto fonte, se ter inserido o seguinte código **Haskell**:

```
import Cp
import List
```

---

<sup>1</sup>O suffixo ‘lhs’ quer dizer *literate Haskell*.

```

import Nat
import Exp
import BTree
import LTree
import St
import Probability hiding (· → ·, ·)
import Data.List
import Test.QuickCheck hiding ((×))
import System.Random hiding (·, ·)
import GHC.IO.Exception
import System.IO.Unsafe
import Data.Maybe
import Data.Either

```

Abra o ficheiro `cp1617t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

```

\begin{code}
...
\end{code}

```

vai ser seleccionado pelo **GHCi** para ser executado.

### 3 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de três alunos. Os detalhes da avaliação (datas para submissão do relatório e sua defesa oral) são os que forem publicados na **página da disciplina** na *internet*. Recomenda-se uma abordagem equilibrada e participativa dos membros do grupo de trabalho por forma a poderem responder às questões que serão colocadas na defesa oral do relatório.

Em que consiste, então, o *relatório* a que se refere o parágrafo anterior? É a edição do texto que está a ser lido, preenchendo o anexo **C** com as suas respostas. O relatório deverá conter ainda a identificação dos membros do grupo de trabalho, no local respectivo da folha de rosto.

Para gerar o PDF integral do relatório deve-se ainda correr os comando seguintes, que actualizam a bibliografia (com **BibTeX**) e o índice remissivo (com **makeindex**),

```

bibtex cp1617t.aux
makeindex cp1617t.idx

```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário **QuickCheck**<sup>2</sup> que ajuda a validar programas em **Haskell**.

### Problema 1

O controlador de um processo físico baseia-se em dezenas de sensores que enviam as suas leituras para um sistema central, onde é feito o respectivo processamento.

Verificando-se que o sistema central está muito sobrecarregado, surgiu a ideia de equipar cada sensor com um microcontrolador que faça algum pré-processamento das leituras antes de as enviar ao sistema central. Esse tratamento envolve as operações (em vírgula flutuante) de soma, subtracção, multiplicação e divisão.

Há, contudo, uma dificuldade: o código da divisão não cabe na memória do microcontrolador, e não se pretende investir em novos microcontroladores devido à sua elevada quantidade e preço.

Olhando para o código a replicar pelos microcontroladores, alguém verificou que a divisão só é usada para calcular inversos,  $\frac{1}{x}$ . Calibrando os sensores foi possível garantir que os valores a inverter estão entre  $1 < x < 2$ , podendo-se então recorrer à **série de Maclaurin**

$$\frac{1}{x} = \sum_{i=0}^{\infty} (1-x)^i$$

para calcular  $\frac{1}{x}$  sem fazer divisões. Seja então

<sup>2</sup>Para uma breve introdução ver e.g. <https://en.wikipedia.org/wiki/QuickCheck>.

$$\text{inv } x \ n = \sum_{i=0}^n (1-x)^i$$

a função que aproxima  $\frac{1}{x}$  com  $n$  iterações da série de MacLaurin. Mostre que  $\text{inv } x$  é um ciclo-for, implementando-o em Haskell (e opcionalmente em C). Deverá ainda apresentar testes em **QuickCheck** que verifiquem o funcionamento da sua solução. (**Sugestão:** inspire-se no problema semelhante relativo à função  $ns$  da secção 3.16 dos apontamentos [4].)

## Problema 2

Se digitar **man wc** na shell do Unix (Linux) obterá:

```
NAME
    wc -- word, line, character, and byte count

SYNOPSIS
    wc [-clmw] [file ...]

DESCRIPTION
    The wc utility displays the number of lines, words, and bytes contained in
    each input file, or standard input (if no file is specified) to the stan-
    dard output. A line is defined as a string of characters delimited by a
    <newline> character. Characters beyond the final <newline> character will
    not be included in the line count.
    (...)
    The following options are available:
    (...)
        -w    The number of words in each input file is written to the standard
              output.
    (...)

```

Se olharmos para o código da função que, em C, implementa esta funcionalidade [2] e nos focarmos apenas na parte que implementa a opção `-w`, verificamos que a poderíamos escrever, em Haskell, da forma seguinte:

```
wc_w :: [Char] -> Int
wc_w [] = 0
wc_w (c:l) =
  if not (sep c) & lookahead_sep l
  then wc_w l + 1
  else wc_w l
  where
    sep c = (c == ' ' || c == '\n' || c == '\t')
    lookahead_sep [] = True
    lookahead_sep (c:l) = sep c

```

Re-implemente esta função segundo o modelo *worker/wrapper* onde *wrapper* deverá ser um catamorfismo de listas. Apresente os cálculos que fez para chegar a essa sua versão de `wc_w` e inclua testes em **QuickCheck** que verifiquem o funcionamento da sua solução. (**Sugestão:** aplique a lei de recursividade múltipla às funções `wc_w` e `lookahead_sep`.)

## Problema 3

Uma “**B-tree**” é uma generalização das árvores binárias do módulo **BTree** a mais do que duas sub-árvores por nó:

```
data B-tree a = Nil | Block { leftmost :: B-tree a, block :: [(a, B-tree a)] } deriving (Show, Eq)
```

Por exemplo, a B-tree<sup>3</sup>

<sup>3</sup>Créditos: figura extraída de <https://en.wikipedia.org/wiki/B-tree>.



é representada no tipo acima por:

```

t = Block {
  leftmost = Block {
    leftmost = Nil,
    block = [(1, Nil), (2, Nil), (5, Nil), (6, Nil)]},
  block = [
    (7, Block {
      leftmost = Nil,
      block = [(9, Nil), (12, Nil)]}),
    (16, Block {
      leftmost = Nil,
      block = [(18, Nil), (21, Nil)]})
  ]}
  
```

Pretende-se, neste problema:

1. Construir uma biblioteca para o tipo B-tree da forma habitual (in + out; ana + cata + hylo; instância na classe *Functor*).
2. Definir como um catamorfismo a função *inordB\_tree* :: B-tree *t* → [*t*] que faça travessias “in-order” de árvores deste tipo.
3. Definir como um catamorfismo a função *largestBlock* :: B-tree *a* → *Int* que detecta o tamanho do maior bloco da árvore argumento.
4. Definir como um anamorfismo a função *mirrorB\_tree* :: B-tree *a* → B-tree *a* que roda a árvore argumento de 180°
5. Adaptar ao tipo B-tree o hilomorfismo “quick sort” do módulo BTree. O respectivo anamorfismo deverá basear-se no gene *lsplitB\_tree* cujo funcionamento se sugere a seguir:

```

lsplitB_tree [] = i1 ()
lsplitB_tree [7] = i2 ([], [(7, [])])
lsplitB_tree [5, 7, 1, 9] = i2 ([1], [(5, []), (7, [9])])
lsplitB_tree [7, 5, 1, 9] = i2 ([1], [(5, []), (7, [9])])
  
```

6. A biblioteca **Exp** permite representar árvores-expressão em formato DOT, que pode ser lido por aplicações como por exemplo **Graphviz**, produzindo as respectivas imagens. Por exemplo, para o caso de árvores **BTree**, se definirmos

```

dotBTree :: Show a => BTree a → IO ExitCode
dotBTree = dotpict · bmap nothing (Just · show) · cBTree2Exp
  
```

executando *dotBTree* *t* para

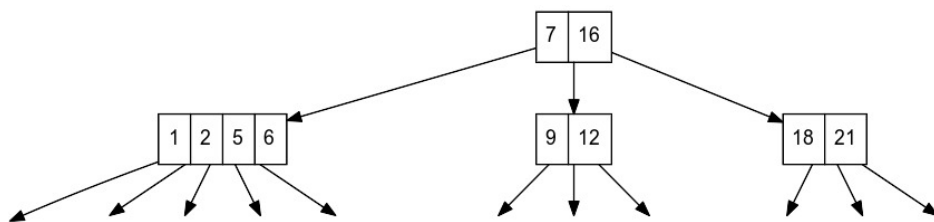
```

t = Node (6, (Node (3, (Node (2, (Empty, Empty)), Empty)), Node (7, (Empty, Node (9, (Empty, Empty))))))
  
```

obter-se-á a imagem



Escreva de forma semelhante uma função `dotB_tree` que permita mostrar em [Graphviz](#)<sup>4</sup> árvores B-tree tal como se ilustra a seguir,



para a árvore dada acima.

## Problema 4

Nesta disciplina estudaram-se funções mutuamente recursivas e como lidar com elas. Os tipos indutivos de dados podem, eles próprios, ser mutuamente recursivos. Um exemplo dessa situação são os chamados **L-Systems**.

Um **L-System** é um conjunto de regras de produção que podem ser usadas para gerar padrões por re-escrita sucessiva, de acordo com essas mesmas regras. Tal como numa gramática, há um axioma ou símbolo inicial, de onde se parte para aplicar as regras. Um exemplo célebre é o do crescimento de algas formalizado por Lindenmayer<sup>5</sup> no sistema:

**Variáveis:**  $A$  e  $B$

**Constantes:** nenhuma

**Axioma:**  $A$

**Regras:**  $A \rightarrow A B, B \rightarrow A$ .

Quer dizer, em cada iteração do “crescimento” da alga, cada  $A$  deriva num par  $A B$  e cada  $B$  converte-se num  $A$ . Assim, ter-se-á, onde  $n$  é o número de iterações desse processo:

- $n = 0$ :  $A$
- $n = 1$ :  $A B$
- $n = 2$ :  $A B A$
- $n = 3$ :  $A B A A B$
- etc

<sup>4</sup>Como alternativa a instalar [Graphviz](#), podem usar [WebGraphviz](#) num browser.

<sup>5</sup>Ver [https://en.wikipedia.org/wiki/Aristid\\_Lindenmayer](https://en.wikipedia.org/wiki/Aristid_Lindenmayer).

Este **L-System** pode codificar-se em Haskell considerando cada variável um tipo, a que se adiciona um caso de paragem para poder expressar as sucessivas iterações:

```
type Algae = A
data A = NA | A A B deriving Show
data B = NB | B A deriving Show
```

Observa-se aqui já que  $A$  e  $B$  são mutuamente recursivos. Os isomorfismos in/out são definidos da forma habitual:

```
inA :: 1 + A × B → A
inA = [NA, A]
outA :: A → 1 + A × B
outA NA = i1 ()
outA (A a b) = i2 (a, b)
inB :: 1 + A → B
inB = [NB, B]
outB :: B → 1 + A
outB NB = i1 ()
outB (B a) = i2 a
```

O functor é, em ambos os casos,  $F X = 1 + X$ . Contudo, os catamorfismos de  $A$  têm de ser estendidos com mais um gene, de forma a processar também os  $B$ ,

$$\begin{aligned} \llbracket \cdot \cdot \rrbracket_A &:: (1 + c \times d \rightarrow c) \rightarrow (1 + c \rightarrow d) \rightarrow A \rightarrow c \\ \llbracket ga \ gb \rrbracket_A &= ga \cdot (id + \llbracket ga \ gb \rrbracket_A \times \llbracket ga \ gb \rrbracket_B) \cdot outA \end{aligned}$$

e a mesma coisa para os  $B$ s:

$$\begin{aligned} \llbracket \cdot \cdot \rrbracket_B &:: (1 + c \times d \rightarrow c) \rightarrow (1 + c \rightarrow d) \rightarrow B \rightarrow d \\ \llbracket ga \ gb \rrbracket_B &= gb \cdot (id + \llbracket ga \ gb \rrbracket_A) \cdot outB \end{aligned}$$

Pretende-se, neste problema:

1. A definição dos anamorfismos dos tipos  $A$  e  $B$ .
2. A definição da função

$$generateAlgae :: Int \rightarrow Algae$$

como anamorfismo de  $Algae$  e da função

$$showAlgae :: Algae \rightarrow String$$

como catamorfismo de  $Algae$ .

3. Use **QuickCheck** para verificar a seguinte propriedade:

$$length \cdot showAlgae \cdot generateAlgae = fib \cdot succ$$

## Problema 5

O ponto de partida deste problema é um conjunto de equipas de futebol, por exemplo:

```
equipas :: [Equipa]
equipas = [
  "Arouca", "Belenenses", "Benfica", "Braga", "Chaves", "Feirense",
  "Guimaraes", "Maritimo", "Moreirense", "Nacional", "P.Ferreira",
  "Porto", "Rio Ave", "Setubal", "Sporting", "Estoril"
]
```

Assume-se que há uma função  $f(e_1, e_2)$  que dá — baseando-se em informação acumulada historicamente, e.g. estatística — qual a probabilidade de  $e_1$  ou  $e_2$  ganharem um jogo entre si.<sup>6</sup> Por exemplo,  $f(\text{"Arouca"}, \text{"Braga"})$  poderá dar como resultado a distribuição

Arouca     28.6%  
 Braga     71.4%

indicando que há 71.4% de probabilidades de "Braga" ganhar a "Arouca".

Para lidarmos com probabilidades vamos usar o mónade  $\text{Dist } a$  que vem descrito no apêndice A e que está implementado na biblioteca **Probability** [1] — ver definição (1) mais adiante. A primeira parte do problema consiste em sortear *aleatoriamente* os jogos das equipas. O resultado deverá ser uma **LTree** contendo, nas folhas, os jogos da primeira eliminatória e cujos nós indicam quem joga com quem (vencendo), à medida que a eliminatória prossegue:



A segunda parte do problema consiste em processar essa árvore usando a função

$jogo :: (Equipa, Equipa) \rightarrow \text{Dist } Equipa$

que foi referida acima. Essa função simula um qualquer jogo, como foi acima dito, dando o resultado de forma probabilística. Por exemplo, para o sorteio acima e a função *jogo* que é dada neste enunciado<sup>7</sup>, a probabilidade de cada equipa vir a ganhar a competição vem dada na distribuição seguinte:

Porto             21.7%  
 Sporting        21.4%  
 Benfica        19.0%  
 Guimaraes     9.4%  
 Braga         5.1%  
 Nacional       4.9%  
 Maritimo      4.1%  
 Belenenses    3.5%  
 Rio Ave       2.3%  
 Moreirense    1.9%  
 P.Ferreira    1.4%  
 Arouca        1.4%  
 Estoril       1.4%  
 Setubal       1.4%  
 Feirense      0.7%  
 Chaves        0.4%

Assumindo como dada e fixa a função *jogo* acima referida, juntando as duas partes obteremos um *hilomorfismo* de tipo  $[Equipa] \rightarrow \text{Dist } Equipa$ ,

$quem\_vence :: [Equipa] \rightarrow \text{Dist } Equipa$   
 $quem\_vence = eliminatória \cdot sorteio$

com características especiais: é aleatório no anamorfismo (sorteio) e probabilístico no catamorfismo (eliminatória).

<sup>6</sup>Tratando-se de jogos eliminatórios, não há lugar a empates.

<sup>7</sup>Pode, se desejar, criar a sua própria função *jogo*, mas para efeitos de avaliação terá que ser usada a que vem dada neste enunciado. Uma versão de *jogo* realista teria que ter em conta todas as estatísticas de jogos entre as equipas em jogo, etc etc.



O anamorfismo  $\text{sorteio} :: [Equipa] \rightarrow \text{LTree } Equipa$  tem a seguinte arquitectura,<sup>8</sup>

$$\text{sorteio} = \text{anaLTree } \text{lsplit} \cdot \text{envia} \cdot \text{permuta}$$

reutilizando o anamorfismo do algoritmo de “merge sort”, da biblioteca **LTree**, para construir a árvore de jogos a partir de uma permutação aleatória das equipas gerada pela função genérica

$$\text{permuta} :: [a] \rightarrow \text{IO } [a]$$

A presença do mónade de IO tem a ver com a geração de números aleatórios<sup>9</sup>.

1. Defina a função monádica  $\text{permuta}$  sabendo que tem já disponível

$$\text{getR} :: [a] \rightarrow \text{IO } (a, [a])$$

$\text{getR } x$  dá como resultado um par  $(h, t)$  em que  $h$  é um elemento de  $x$  tirado à sorte e  $t$  é a lista sem esse elemento – mas esse par vem encapsulado dentro de IO.

2. A segunda parte do exercício consiste em definir a função monádica

$$\text{eliminatória} :: \text{LTree } Equipa \rightarrow \text{Dist } Equipa$$

que, assumindo já disponível a função  $\text{jogo}$  acima referida, dá como resultado a distribuição de equipas vencedoras do campeonato.

**Sugestão:** inspire-se na secção 4.10 (*‘Monadification’ of Haskell code made easy*) dos apontamentos [4].

## Referências

- [1] M. Erwig and S. Kollmansberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16:21–34, January 2006.
- [2] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, N.J., 1978.
- [3] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.
- [4] J.N. Oliveira. *Program Design by Calculation*, 2008. Draft of textbook in preparation. viii+297 pages. Informatics Department, University of Minho.

---

<sup>8</sup>A função  $\text{envia}$  não é importante para o processo; apenas se destina a simplificar a arquitectura monádica da solução.

<sup>9</sup>Quem estiver interessado em detalhes deverá consultar **System.Random**.



## B Definições auxiliares

São dadas: a função que simula jogos entre equipas,

```
type Equipa = String
jogo :: (Equipa, Equipa) → Dist Equipa
jogo (e1, e2) = D [(e1, 1 - r1 / (r1 + r2)), (e2, 1 - r2 / (r1 + r2))] where
  r1 = rank e1
  r2 = rank e2
  rank = pap ranks
  ranks = [
    ("Arouca", 5),
    ("Belenenses", 3),
    ("Benfica", 1),
    ("Braga", 2),
    ("Chaves", 5),
    ("Feirense", 5),
    ("Guimaraes", 2),
    ("Maritimo", 3),
    ("Moreirense", 4),
    ("Nacional", 3),
    ("P.Ferreira", 3),
    ("Porto", 1),
    ("Rio Ave", 4),
    ("Setubal", 4),
    ("Sporting", 1),
    ("Estoril", 5)]
```

a função (monádica) que parte uma lista numa cabeça e cauda *aleatórias*,

```
getR :: [a] → IO (a, [a])
getR x = do {
  i ← getStdRandom (randomR (0, length x - 1));
  return (x !! i, retira i x)
} where retira i x = take i x ++ drop (i + 1) x
```

e algumas funções auxiliares de menor importância: uma que ordena listas com base num atributo (função que induz uma pré-ordem),

```
presort :: (Ord a, Ord b) ⇒ (b → a) → [b] → [b]
presort f = map π2 · sort · (map (fork f id))
```

e outra que converte “look-up tables” em funções (parciais):

```
pap :: Eq a ⇒ [(a, t)] → a → t
pap m k = unJust (lookup k m) where unJust (Just a) = a
```

## C Soluções propostas

Os alunos devem colocar neste anexo as suas soluções aos exercícios propostos, de acordo com o “layout” que se fornece. Não podem ser alterados os nomes das funções dadas, mas pode ser adicionado texto e / ou outras funções auxiliares que sejam necessárias.

### Problema 1

```
inv x = π2 · (for ⟨((1 - x)*) · π1, (⊕) · (((1 - x)*) × id)⟩ (1, 1))
prop.inv x = (x > 1 ∧ x < 2) ==> abs ((inv (inv x 5000) 5000) - x) < diferenca
  where diferenca = 0.000000009
```

Inversa antes da conversão de uma catamorfismo de naturais para um for

```

inv x = p2.cataNat (y)
  where
    y = split z p
    p = either (const 1) ((uncurry(+)).((1-x)*) >< id)
    z = either (const 1) (((1-x)*)).(id + p1)

```

Outra resolução da inversa através do seguinte raciocínio :

```

inv x = for ((1+) . ((1-x)*)) 1

```

Prova:

$$\begin{aligned}
& \left\{ \begin{array}{l} inv\ x\ 0 = 1 \\ inv\ x\ (n+1) = (1-x) \uparrow (n+1) \end{array} \right. \\
& = \{ \text{lookaheap=look ; def sep c ; []=nil ; def cons ; lei-81} \} \\
& \left\{ \begin{array}{l} look \cdot nil = true \\ look \cdot cons\ (c, l) = sep \cdot \pi_1 \cdot (c, look) \end{array} \right. \\
& = \{ \text{lei-73} \} \\
& \left\{ \begin{array}{l} look \cdot nil = true \\ look \cdot cons = sep \cdot \pi_1 \end{array} \right. \\
& = \{ \text{lei-17 ; in lista=[nil, cons] ; lei-76} \} \\
& look \cdot \mathbf{in} = [true, sep \cdot \pi_1] \\
& = \{ \text{lei-1} \} \\
& look \cdot \mathbf{in} = [true \cdot id, sep \cdot id \cdot \pi_1] \\
& = \{ \text{lei-13} \} \\
& look \cdot \mathbf{in} = [true \cdot id, sep \cdot \pi_1 \cdot (id \times \langle look, wc \rangle)] \\
& = \{ \text{lei-22 (Resultado 1)} \} \\
& look \cdot \mathbf{in} = [true, sep \cdot \pi_1] \cdot (id + (id \times \langle look, wc \rangle)) \\
& = \{ \text{Por Fokking} \} \\
& wc\_c \cdot \mathbf{in} = [f, g] \cdot (id + (id \times \langle look, wc \rangle)) \\
& = \{ \text{Lei-22 ; lei-17} \} \\
& \left\{ \begin{array}{l} wc\_c \cdot nil = f \\ wc\_c \cdot cons = g \cdot (id \times \langle look, wc \rangle) \end{array} \right. \\
& = \{ f=0 ; lei\ 73 \} \\
& \left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons\ (c, l) = g \cdot (id \times \langle look, wc \rangle)\ (c, l) \end{array} \right. \\
& = \{ \text{lei-79} \} \\
& \left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons\ (c, l) = g \cdot (id\ c \times \langle look, wc \rangle\ l) \end{array} \right. \\
& = \{ \text{lei-78 ; lei\ 75} \} \\
& \left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons\ (c, l) = g \cdot (c \times \langle look\ l, wc\ l \rangle) \end{array} \right. \\
& = \{ \text{Atendendo ao código Pointwise} \}
\end{aligned}$$

$$\begin{aligned}
& \left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons(c, l) = ((\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)) \cdot (c \times (look\ l, wc\ l)) \end{array} \right\} \\
& = \{ \text{Lei 17} \} \\
& wc\_c \cdot [nil, cons(c, l)] = [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (c \times (look\ l, wc\ l)) \\
& = \{ \text{encontrado o } [f, g]; (\text{resultado 2}) \} \\
& wc\_c \cdot \mathbf{in} = [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (id + (id \times \langle look, wc \rangle)) \\
& = \{ \text{Aplicando Fokking ao resultado 1 e 2} \} \\
& \langle wc\_c, l \rangle = (\langle \langle \underline{true}, sep \cdot \pi_1 \rangle, [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \rangle) \\
& = \{ \text{lei 7 ; lei-28} \} \\
& wc\_c = \pi_2 \cdot (\langle \langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle \rangle) \\
& = \{ \text{uncurry ; lei 78} \} \\
& wc\_c = \pi_2 \cdot (\langle \langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\widehat{(\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2)} \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)) \rangle \rangle) \\
& = \{ \text{lei-8 ; lei-78} \} \\
& wc\_c = \pi_2 \cdot (\langle \langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\widehat{(\neg \cdot sep \times \pi_1)}) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle \rangle)
\end{aligned}$$

## Problema 2

$$\begin{aligned}
& wc\_w\_final :: [Char] \rightarrow Int \\
& wc\_w\_final = wrapper \cdot worker \\
& wrapper = \pi_2 \\
& worker = cataList [\langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\widehat{(\neg \cdot sep)} \times (\pi_1)) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle] \\
& \text{where} \\
& \quad sep\ c = c \equiv ' ' \vee c \equiv ' \backslash n' \vee c \equiv ' \backslash t'
\end{aligned}$$

A resolução deste exercício consistiu em duas partes: Encontrar o worker e wrapper utilizando a lei da recursividade múltipla e cálculo de programas.

$$\begin{aligned}
& \left\{ \begin{array}{l} lookahead [] = true \\ lookahead (c : l) = (c \equiv ' ' \mid c \equiv ' \backslash n' \mid c \equiv ' \backslash t' ) \end{array} \right\} \\
& = \{ \text{lookahead=look ; def sep c ; []=nil ; def cons ; lei-81} \} \\
& \left\{ \begin{array}{l} look \cdot nil = true \\ look \cdot cons(c, l) = sep \cdot \pi_1 \cdot (c, look) \end{array} \right\} \\
& = \{ \text{lei-73} \} \\
& \left\{ \begin{array}{l} look \cdot nil = true \\ look \cdot cons = sep \cdot \pi_1 \end{array} \right\} \\
& = \{ \text{lei-17 ; in lista=[nil, cons] ; lei-76} \} \\
& look \cdot \mathbf{in} = [\underline{true}, sep \cdot \pi_1] \\
& = \{ \text{lei-1} \} \\
& look \cdot \mathbf{in} = [\underline{true} \cdot id, sep \cdot id \cdot \pi_1] \\
& = \{ \text{lei-13} \} \\
& look \cdot \mathbf{in} = [\underline{true} \cdot id, sep \cdot \pi_1 \cdot (id \times \langle look, wc \rangle)] \\
& = \{ \text{lei-22 (Resultado 1)} \} \\
& look \cdot \mathbf{in} = [\underline{true}, sep \cdot \pi_1] \cdot (id + (id \times \langle look, wc \rangle))
\end{aligned}$$

$$\begin{aligned}
&= \{ \text{Por Fokking} \} \\
wc\_c \cdot \mathbf{in} &= [f, g] \cdot (id + (id \times \langle look, wc \rangle)) \\
&= \{ \text{Lei-22 ; lei-17} \} \\
&\left\{ \begin{array}{l} wc\_c \cdot nil = f \\ wc\_c \cdot cons = g \cdot (id \times \langle look, wc \rangle) \end{array} \right. \\
&= \{ f=0 ; lei 73 \} \\
&\left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons (c, l) = g \cdot (id \times \langle look, wc \rangle) (c, l) \end{array} \right. \\
&= \{ lei-79 \} \\
&\left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons (c, l) = g \cdot (id \times c \times \langle look, wc \rangle l) \end{array} \right. \\
&= \{ lei-78 ; lei 75 \} \\
&\left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons (c, l) = g \cdot (c \times (look l, wc l)) \end{array} \right. \\
&= \{ \text{Atendendo ao código Pointwise} \} \\
&\left\{ \begin{array}{l} wc\_c \cdot nil = 0 \\ wc\_c \cdot cons (c, l) = ((\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)) \cdot (c \times (look l, wc l)) \end{array} \right. \\
&= \{ \text{Lei 17} \} \\
wc\_c \cdot [nil, cons (c, l)] &= [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (c \times (look l, wc l)) \\
&= \{ \text{encontrado o } [f, g]; (\text{resultado 2}) \} \\
wc\_c \cdot \mathbf{in} &= [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (id + (id \times \langle look, wc \rangle)) \\
&= \{ \text{Aplicando Fokking ao resultado 1 e 2} \} \\
\langle wc\_c, l \rangle &= (\langle [\underline{true}, sep \cdot \pi_1], [0, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \rangle) \\
&= \{ lei 7 ; lei-28 \} \\
wc\_c &= \pi_2 \cdot \langle [\langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\neg \cdot sep \cdot \pi_1 \wedge \pi_1 \cdot \pi_2) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle] \rangle \\
&= \{ \text{uncurry ; lei 78} \} \\
wc\_c &= \pi_2 \cdot \langle [\langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\widehat{(\neg \cdot sep \cdot \pi_1, \pi_1 \cdot \pi_2)}) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle] \rangle \\
&= \{ lei-8 ; lei-78 \} \\
wc\_c &= \pi_2 \cdot \langle [\langle \underline{true}, 0 \rangle, \langle sep \cdot \pi_1, (\widehat{(\neg \cdot sep \times \pi_1)}) \rightarrow (succ \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle] \rangle
\end{aligned}$$

QuickCheck:

```

genSafeChar :: Gen Char
genSafeChar = elements ([ 'a' .. 'z' ] ++ " \n\t ")

genSafeString :: Gen String
genSafeString = listOf genSafeChar

newtype SafeString = SafeString { unwrapSafeString :: String }
deriving Show

prop_wc_w_final = forAll genSafeString $ \str -> (wc_w_final str) == (length $ words str)

```

Consulta de:

<https://stackoverflow.com/questions/20934506/haskell-quickcheck-how-to-generate-on>  
in=(either (const 0) succ); f=inv; g=macL; F(split f g)=F(split inv macL)=( id + (split inv macL))

### Problema 3

-- (1)-----  
outB\_tree :: B-tree a -> () + (B-tree a, [(a, B-tree a)])

```

outB_tree Nil = i1 ()
outB_tree (Block t l) = i2 (t, l)
inB_tree :: () + (B-tree a, [(a, B-tree a)]) → B-tree a
inB_tree = [Nil, Block]
baseB_tree :: (a → b) → (c → d) → () + (c, [(a, c)]) → () + (d, [(b, d)])
baseB_tree f g = id + (g × (map (f × g)))
recB_tree :: (a → b) → () + (a, [(d, a)]) → () + (b, [(d, b)])
recB_tree g = baseB_tree id g
cataB_tree :: ((() + (b, [(a, b)])) → b) → B-tree a → b
cataB_tree g = g · (recB_tree (cataB_tree g)) · outB_tree
anaB_tree :: (a → () + (a, [(b, a)])) → a → B-tree b
anaB_tree g = inB_tree · (recB_tree (anaB_tree g)) · g
hyloB_tree :: ((() + (c, [(b, c)])) → c) → (a → () + (a, [(b, a)])) → a → c
hyloB_tree h g = cataB_tree h · anaB_tree g
instance Functor B-tree
  where fmap f = cataB_tree (inB_tree · baseB_tree f id)
-- (2)

```

Diagrama da função *inordB\_tree*:

$$\begin{array}{ccc}
\text{B-tree } a & \xrightarrow{\langle g \rangle = \text{inordB\_tree}} & L \\
\text{outB\_tree} \downarrow & & \uparrow g \\
1 + \text{B-tree } a \times (a \times \text{B-tree } a) & \xrightarrow{id + \langle g \rangle \times \text{map } (id \times \langle g \rangle)} & 1 + L \times (a \times L)
\end{array}$$

```

inordB_tree :: B-tree t → [t]
inordB_tree = cataB_tree inordB_tree_gene
inordB_tree_gene :: () + ([a], [(a, [a])]) → [a]
inordB_tree_gene = [nil, conc · (id × (concat · (map cons)))]
-- (3)

```

Diagrama da função *largestBlock*:

$$\begin{array}{ccc}
\text{B-tree } a & \xrightarrow{\langle g \rangle = \text{largestBlock}} & \mathbb{N}_0 \\
\text{outB\_tree} \downarrow & & \uparrow g \\
1 + \text{B-tree } a \times (a \times \text{B-tree } a) & \xrightarrow{id + \langle g \rangle \times \text{map } (id \times \langle g \rangle)} & 1 + \mathbb{N}_0 \times (a \times \mathbb{N}_0)
\end{array}$$

```

largestBlock :: B-tree a → Int
largestBlock = cataB_tree [0, max · (id × (max · ⟨length, maximum · (map π2)⟩))]
-- (4)

```

```

mirrorB_tree :: B-tree a → B-tree a
mirrorB_tree = anaB_tree g
  where g = (id + (toTuple · reverse · toEither)) · outB_tree
toTuple :: [a + b] → (b, [(a, b)])
toTuple = ⟨head · π2, zip · (id × tail)⟩ · partitionEithers
toEither :: (b, [(a, b)]) → [a + b]
toEither = cons · (i2 × (concat · map listTuple))
listTuple :: (a, b) → [a + b]
listTuple = cons · (i1 × (cons · ⟨i2, []⟩))
-- (5)

```

```

qSortB_tree :: Ord a => [a] -> [a]
qSortB_tree = hyloB_tree inordB_tree_gene lsplitB_tree
lsplitB_tree :: Ord a => [a] -> () + ([a], [(a, [a])])
lsplitB_tree [] = i1 ()
lsplitB_tree (h : t) = i2 (l, [(h, r)])
  where (l, r) = mypart (<h) t

```

Diagrama da função *mypart*:

$$\begin{array}{ccc}
 L & \xrightarrow{\quad \langle g \rangle = \text{mypart } p \quad} & (L1)^* \times (L2)^* \\
 \text{outList} \downarrow & & \uparrow g \\
 1 + a \times L & \xrightarrow{\quad id + id \times \langle g \rangle \quad} & 1 + a \times ((L1)^* \times (L2)^*)
 \end{array}$$

```

mypart :: (a -> Bool) -> [a] -> ([a], [a])
mypart p = cataList [(nil, nil), (p · π1) -> toFST, toSND]
toFST :: (a, ([a], [a])) -> ([a], [a])
toFST = <cons · (id × π1), π2 · π2>
toSND :: (a, ([a], [a])) -> ([a], [a])
toSND = <π1 · π2, cons · (id × π2)>
-- (6)-----
dotB_tree :: Show a => B-tree a -> IO ExitCode
dotB_tree = dotpict · bmap nothing (Just · show) · cB_tree2Exp
cB_tree2Exp :: B-tree a -> Exp [Char] [a]
cB_tree2Exp = cataB_tree [· $ Var "nil", aux]
  where aux = Term · ((map π1) · π2, cons · (id × ((map π2))))

```

#### Problema 4

```

[[ga gb]]A = inA · ((id) + ([[ga gb]]A × [[ga gb]]B)) · ga
[[ga gb]]B = inB · ((id) + [[ga gb]]A) · gb

generateAlgae = [[(id + ⟨id, id⟩) · outNat outNat]]A
showAlgae = [[["A", conc · id] ["B", id]]]A
prop_Algae :: Int -> Property
prop_Algae x = (x ≥ 0 ∧ x < 20) ==> (a x) ≡ (b x)
  where
    a = length · showAlgae · generateAlgae
    b = fromInteger · fib · succ · toInteger

```

#### Problema 5

```

permuta = ⊥
eliminatoria = ⊥

```



# Índice

- LaTeX, 2
  - lhs2TeX, 2
- B-tree, 4
- Cálculo de Programas, 3
  - Material Pedagógico, 2
    - BTree.hs, 4, 5
    - Exp.hs, 5
    - LTree.hs, 8, 9
- Combinador “pointfree”
  - cata*, 7
  - either*, 7
- Função
  - $\pi_2$ , 11
  - length*, 7, 11
  - map*, 11
  - succ*, 7
  - uncurry*, 7
- Functor, 3, 5, 7–11
- Graphviz, 5, 6
  - WebGraphviz, 6
- Haskell, 2, 3
  - “Literate Haskell”, 2
  - Biblioteca
    - PFP, 10
    - Probability, 8, 10
  - interpretador
    - GHCI, 3, 10
  - QuickCheck, 3, 4, 7
- L-system, 6, 7
- Programação literária, 2
- Taylor series
  - Maclaurin series, 3
- U.Minho
  - Departamento de Informática, 1
- Unix shell
  - wc*, 4
- Utilitário
  - LaTeX
    - bibtex*, 3
    - makeindex*, 3