



**Universidade do Minho**  
Escola de Engenharia

## **Cálculo de Programas**

### Trabalho Prático (2023/24)

Lic. em Engenharia Informática

#### **Grupo G99**

xxxxxxx	Nome
xxxxxxx	Nome
xxxxxxx	Nome

# Preâmbulo

**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 corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos cursos que têm esta disciplina, opta-se pela aplicação deste método à programação em **Haskell** (sem prejuízo da sua aplicação a outras linguagens funcionais). Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em **Haskell**. Há ainda um outro objectivo: o de ensinar a documentar programas, a validá-los e a produzir textos técnico-científicos de qualidade.

Antes de abordarem os problemas propostos no trabalho, os grupos devem ler com atenção o anexo **A** onde encontrarão as instruções relativas ao software a instalar, etc.

Valoriza-se a escrita de *pouco* código que corresponda a soluções simples e elegantes que utilizem os combinadores de ordem superior estudados na disciplina.

## 1 Exercício 1

Este problema, retirado de um *site* de exercícios de preparação para entrevistas de emprego, tem uma formulação simples:

*Dada uma matriz de uma qualquer dimensão, listar todos os seus elementos rodados em espiral.*

*Por exemplo, dadas as seguintes matrizes:*

1	→	2	→	3
				↓
4	→	5		6
↑				↓
7	←	8	←	9

1	→	2	→	3	→	4
						↓
5	→	6	→	7		8
↑						↓
9	←	10	←	11	←	12

*dever-se-á obter, respetivamente,  $[1, 2, 3, 6, 9, 8, 7, 4, 5]$  e  $[1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]$ .*

□

Valorizar-se-ão as soluções *pointfree* que empreguem os combinadores estudados na disciplina, e.g.  $f \cdot g$ ,  $\langle f, g \rangle$ ,  $f \times g$ ,  $[f, g]$ ,  $f + g$ , bem como catamorfismos e anamorfismos.

Recomenda-se a escrita de *pouco* código e de soluções simples e fáceis de entender. Recomenda-se que o código venha acompanhado de uma descrição de como funciona e foi concebido, apoiado em diagramas explicativos. Para instruções sobre como produzir esses diagramas e exprimir raciocínios de cálculo, ver o anexo **D**.

## 1.1 Resolução

De maneira a resolver este problema decidimos fazer alguma pesquisa. Chegamos à conclusão que este problema era bastante conhecido como **matrot**. A nossa resolução baseia-se na utilização das funções *transpose* e *reverse*.

### Problema 1

Este problema, que de novo foi retirado de um *site* de exercícios de preparação para entrevistas de emprego, tem uma formulação muito simples:

*Inverter as vogais de um string.*

Esta formulação deverá ser generalizada a:

*Inverter os elementos de uma dada lista que satisfazem um dado predicado.*

Valorizam-se as soluções tal como no problema anterior e fazem-se as mesmas recomendações.

### Problema 2

A fornecer na segunda edição deste enunciado

### Problema 3

A fornecer na segunda edição deste enunciado

## Anexos

### A Natureza do trabalho a realizar

Este trabalho teórico-prático deve ser realizado por grupos de 3 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 participativa dos membros do grupo em **todos** os exercícios do trabalho, para assim poderem responder a qualquer questão colocada na *defesa oral* do relatório.

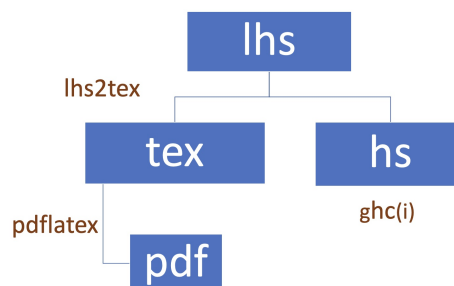
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita “[literária](#)” [1], cujo princípio base é o seguinte:

*Um programa e a sua documentação devem coincidir.*

Por outras palavras, o **código fonte** e a **documentação** de um programa deverão estar no mesmo ficheiro.

O ficheiro `cp2324t.pdf` que está a ler é já um exemplo de [programação literária](#): foi gerado a partir do texto fonte `cp2324t.lhs`<sup>1</sup> que encontrará no [material pedagógico](#) desta disciplina descompactando o ficheiro `cp2324t.zip`.

Como se mostra no esquema abaixo, de um único ficheiro (*lhs*) gera-se um PDF ou faz-se a interpretação do código [Haskell](#) que ele inclui:



Vê-se assim que, para além do [GHCi](#), serão necessários os executáveis [pdflatex](#) e [lhs2TeX](#). Para facilitar a instalação e evitar problemas de versões e conflitos com sistemas operativos, é recomendado o uso do [Docker](#) tal como a seguir se descreve.

### B Docker

Recomenda-se o uso de um [container](#) Docker que deverá ser gerado a partir do ficheiro `Dockerfile` que se encontra na diretoria que resulta de descompactar `cp2324t.zip`. Este [container](#) deverá ser usado na execução do [GHCi](#) e dos comandos relativos ao [L<sup>A</sup>T<sub>E</sub>X](#). (Ver também a `Makefile` que é disponibilizada.)

Após [instalar o Docker](#) e descarregar o referido zip com o código fonte do trabalho, basta executar os seguintes comandos:

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

```
$ docker build -t cp2324t .
$ docker run -v ${PWD}:/cp2324t -it cp2324t
```

**NB:** O objetivo é que o container seja usado *apenas* para executar o [GHCi](#) e os comandos relativos ao [L<sup>A</sup>T<sub>E</sub>X](#). Deste modo, é criado um *volume* (cf. a opção `-v ${PWD}:/cp2324t`) que permite que a diretoria em que se encontra na sua máquina local e a diretoria `/cp2324t` no [container](#) sejam partilhadas.

Pretende-se então que visualize/edite os ficheiros na sua máquina local e que os compile no [container](#), executando:

```
$ lhs2TeX cp2324t.lhs > cp2324t.tex
$ pdflatex cp2324t
```

[lhs2TeX](#) é o pre-processor que faz “pretty printing” de código Haskell em [L<sup>A</sup>T<sub>E</sub>X](#) e que faz parte já do [container](#). Alternativamente, basta executar

```
$ make
```

para obter o mesmo efeito que acima.

Por outro lado, o mesmo ficheiro `cp2324t.lhs` é executável e contém o “kit” básico, escrito em [Haskell](#), para realizar o trabalho. Basta executar

```
$ ghci cp2324t.lhs
```

Abra o ficheiro `cp2324t.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}
```

é seleccionado pelo [GHCi](#) para ser executado.

## C Em que consiste o TP

Em que consiste, então, o *relatório* a que se referiu acima? É a edição do texto que está a ser lido, preenchendo o anexo [F](#) com as 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 [Bib<sub>T</sub>E<sub>X</sub>](#)) e o índice remissivo (com [makeindex](#)),

```
$ bibtex cp2324t.aux
$ makeindex cp2324t.idx
```

e recompilar o texto como acima se indicou. (Como já se disse, pode fazê-lo correndo simplesmente `make` no [container](#).)

No anexo [E](#) disponibiliza-se algum código [Haskell](#) relativo aos problemas que são colocados. Esse anexo deverá ser consultado e analisado à medida que isso for necessário.

Deve ser feito uso da [programação literária](#) para documentar bem o código que se desenvolver, em particular fazendo diagramas explicativos do que foi feito e tal como se explica no anexo [D](#) que se segue.

## D Como exprimir cálculos e diagramas em LaTeX/lhs2TeX

Como primeiro exemplo, estudar o texto fonte ([lhs](#)) do que está a ler<sup>1</sup> onde se obtém o efeito seguinte:<sup>2</sup>

$$\begin{aligned}
 id &= \langle f, g \rangle \\
 &\equiv \{ \text{universal property} \} \\
 &\quad \left\{ \begin{array}{l} \pi_1 \cdot id = f \\ \pi_2 \cdot id = g \end{array} \right. \\
 &\equiv \{ \text{identity} \} \\
 &\quad \left\{ \begin{array}{l} \pi_1 = f \\ \pi_2 = g \end{array} \right. \\
 &\square
 \end{aligned}$$

Os diagramas podem ser produzidos recorrendo à *package* [xymatrix](#), por exemplo:

$$\begin{array}{ccc}
 \mathbb{N}_0 & \xleftarrow{\text{in}} & 1 + \mathbb{N}_0 \\
 \scriptstyle (g) \downarrow & & \downarrow \scriptstyle id + (g) \\
 B & \xleftarrow{g} & 1 + B
 \end{array}$$

## E Código fornecido

### Problema 1

```

m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
m2 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
m3 = words "Cristina Monteiro Carvalho Sequeira"
test1 = matrot m1 ≡ [1, 2, 3, 6, 9, 8, 7, 4, 5]
test2 = matrot m2 ≡ [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]
test3 = matrot m3 ≡ "CristinaooarieuqeSCMonteirlavra"

```

### Problema 2

```

test4 = reverseVowels "" ≡ ""
test5 = reverseVowels "ácidos" ≡ "ocidás"
test6 = reverseByPredicate even [1..20] ≡ [1, 20, 3, 18, 5, 16, 7, 14, 9, 12, 11, 10, 13, 8, 15, 6, 17, 4, 19, 2]

```

## F Soluções dos alunos

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

<sup>1</sup> Procure e.g. por "sec:diagramas".

<sup>2</sup> Exemplos tirados de [2].

**Importante:** Não pode ser alterado o texto deste ficheiro fora deste anexo.

## Problema 1

$matrot :: Eq\ a \Rightarrow [[a]] \rightarrow [a]$   
 $matrot = \perp$

## Problema 2

$reverseVowels :: String \rightarrow String$   
 $reverseVowels = \perp$   
 $reverseByPredicate :: (a \rightarrow Bool) \rightarrow [a] \rightarrow [a]$   
 $reverseByPredicate\ p = \perp$

# Index

$\LaTeX$ , [3](#), [4](#)

**bibtex**, [4](#)

**lhs2TeX**, [3–5](#)

**makeindex**, [4](#)

**pdflatex**, [3](#)

**xymatrix**, [5](#)

Combinador “pointfree”

*cata*

        Naturais, [5](#)

*either*, [1](#)

*split*, [1](#), [5](#)

Cálculo de Programas, [1](#), [3](#)

    Material Pedagógico, [3](#)

Docker, [3](#)

    container, [3](#), [4](#)

Função

$\pi_1$ , [5](#)

$\pi_2$ , [5](#)

Haskell, [1](#), [3](#), [4](#)

    interpretador

        GHCi, [3](#), [4](#)

    Literate Haskell, [3](#)

Números naturais ( $\mathbb{N}$ ), [5](#)

Programação

    literária, [3](#), [4](#)



## References

- [1] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.
- [2] J.N. Oliveira. *Program Design by Calculation*, 2018. Draft of textbook in preparation. viii+297 pages. Informatics Department, University of Minho.