

# Desenvolvimento de Programas *Haskell*

## Laboratórios de Informática I

### MIEI

Na última sessão tivemos oportunidade de executar um programa de exemplo desenvolvido na linguagem *Haskell*. Para o efeito utilizamos o compilador `ghc`, que produziu um ficheiro executável a partir do código fonte escrito em *Haskell* (no caso, o ficheiro `HelloWorld.hs` criado com o auxílio de um editor de texto).

Recorde-se que o executável produzido pelo compilador avalia uma função particular do programa fornecido — a função `main` do módulo `Main`. Quer isto dizer que outras funções incluídas no programa só serão executadas se forem elas próprias requeridas na avaliação da função `main` (e.g. se forem “invocadas” na função `main`). Este comportamento não é o mais apropriado numa fase de desenvolvimento, quando ainda só se implementou parte da funcionalidade e interessa testar cada função separadamente.

Nesta semana, iremos tomar contacto com uma outra ferramenta de desenvolvimento de programas *Haskell* mais adequada para a fase de desenvolvimento dos programas — o *interpretador*.

## 1 O Interpretador `ghci`

Ao contrário do compilador, o interpretador não produz qualquer ficheiro executável a partir de um programa *Haskell*. Em vez disso, disponibiliza ao programador um ambiente onde pode avaliar qualquer expressão *Haskell* à sua escolha.

O interpretador é invocado pelo comando `ghci`. Uma vez invocado, surge o *prompt* `Prelude>`, sinalizando que o interpretador aguarda um comando do utilizador. Neste ponto pode-se avaliar uma qualquer expressão *Haskell* (e.g.

3+2\*2), ou um comando específico do interpretador (e.g. `:load Fich.hs`, que carrega o ficheiro tornando disponíveis as várias funções aí definidas).

## 1.1 Alguns comandos do ghci

- `:?` ou `:help` — mostra informação sobre comandos do `ghci`;
- `:quit` — sai do interpretador;
- `:cd <dir>` — altera directoria corrente para `<dir>`;
- `:load <mod>` — carrega módulo `<mod>` (ficheiro);
- `:reload` — recarrega último módulo;
- `:type <expr>` — imprime tipo da expressão `<expr>`
- `:info <symb>` — imprime informação sobre símbolo `<symb>`
- `:!<cmd>` — invoca o comando UNIX `<cmd>`

Quando não existir ambiguidade, o `ghci` aceita também abreviaturas dos comandos — por exemplo, o comando `:load HelloWorld` pode simplesmente ser escrito `:l HelloWorld`.

## 2 Utilização de Bibliotecas

A linguagem *Haskell*, tal como a generalidade das linguagens de programação, disponibiliza um conjunto de *bibliotecas* que oferecem ao programador um vasto leque de funcionalidade. Como regra, para utilizar uma biblioteca é necessário *importar* o respectivo módulo. A título de exemplo, no módulo `Data.Char` encontramos funções para manipular valores do tipo `Char` (a representação dos *caracteres*) — para ter acesso a essa funcionalidade é então necessário incluir a declaração `import Data.Char` no início do programa.

## 2.1 Documentação

Um recurso particularmente útil quando recorremos às bibliotecas oferecidas pela linguagem é a sua documentação — é aí que encontramos qual a funcionalidade oferecida (quais os módulos; tipos e funções disponibilizados), assim como uma descrição sumária de cada função (incluindo o seu tipo).

É possível encontrar a documentação da biblioteca *standard* na página de documentação do `ghc` (<http://www.haskell.org/ghc/docs/latest/html/>)<sup>1</sup>. No item *Libraries* encontrará todas as bibliotecas instaladas pelo GHC. Destas, só iremos ter oportunidade de explorar umas poucas, nomeadamente:

- `Prelude` — conjunto de tipos e funções pré-carregados (i.e. não é necessário importar explicitamente qualquer módulo);
- `Data.Char` — funções de manipulação de caracteres;
- `Data.String`: funções de manipulação de *strings* (i.e. sequências de caracteres);
- `Data.List` — funções de manipulação de listas;
- `Data.Maybe` — funções para manipulação do tipo `Maybe`.

Outros apontadores *web* úteis são:

- <http://www.haskell.org/hoogle/>: disponibiliza um mecanismo de busca sobre a documentação das bibliotecas;
- <http://www.haskell.org>: página oficial da linguagem, que inclui apontadores para todo o tipo de documentação sobre a linguagem (em particular, a própria a especificação da linguagem);
- <https://www.fpcomplete.com/school>: tutoriais *online* da linguagem;
- [http://www.compileonline.com/compile\\_haskell\\_online.php](http://www.compileonline.com/compile_haskell_online.php): ambiente *online* para compilar/executar programas *Haskell*;
- <http://hackage.haskell.org>: sítio que agrega contribuições (*packages*) desenvolvidas em *Haskell*.

---

<sup>1</sup>Se instalou a *Haskell Platform*, deverá dispor de uma cópia local desta página.

### 3 Tarefas

1. Crie a directoria `~/LI1` (obs: se já existir a directoria `~/LI1`, remova todo o seu conteúdo)
2. Implemente e teste (no `ghci`) as funções solicitadas no grupo I do caderno de exercícios de Programação Funcional (funções não recursivas — ver anexo A). Crie para o efeito o ficheiro `Aula2.hs` e grave-o na directoria `LI1`.
3. Para cada uma das funções implementadas, adicione a anotação de tipo respectiva.
4. Crie um programa que permita testar algumas das funções codificadas. Utilize como modelo o programa apresentado no anexo B.
5. Remova a directoria `~/LI1` e todo o seu conteúdo.

### A Funções não recursivas (exercícios PF)

Resolva os exercícios da ficha 1 das aulas práticas de Programação Funcional.

### B Programa de Exemplo

```
module Main where

import System.IO

soma :: Int -> Int -> Int
soma x y = x+y

main = do hSetBuffering stdout NoBuffering
          putStr "x = "
          x_inp <- getLine           -- ler linha do terminal
          let x = read x_inp         -- converte String -> Int
          putStr "y = "
          y_inp <- getLine
          let y = read y_inp
```

```
putStrLn ("x + y = " ++ show (soma x y))
```