

Programação Funcional em Haskell

José Romildo Malaquias

BCC222: Programação Funcional

Universidade Federal de Ouro Preto
Departamento de Computação

27 de setembro de 2023

1	Paradigmas de Programação	1-1
1.1	Paradigmas de programação	1-1
1.2	Técnicas e paradigmas de programação	1-2
1.3	Categorias: programação imperativa e declarativa	1-2
1.3.1	Programação imperativa	1-2
1.3.2	Programação declarativa	1-2
1.4	Programação funcional	1-3
1.4.1	Exemplo: quick sort em C	1-3
1.4.2	Exemplo: quick sort em Haskell	1-3
1.5	A Crise do Software	1-4
1.6	Algumas características de Haskell	1-4
1.7	Antecedentes históricos	1-4
1.8	Algumas empresas que usam Haskell	1-7
1.9	Curso online de Haskell	1-7
2	Ambiente de Desenvolvimento Haskell	2-1
2.1	Haskell	2-1
2.2	Instalação do ambiente de desenvolvimento	2-2
2.2.1	Instalação das ferramentas de desenvolvimento em Haskell	2-2
2.3	O ambiente interativo GHCi	2-2
2.4	Bibliotecas	2-6
3	Expressões e Definições	3-1
3.1	Constantes	3-1
3.2	Aplicação de função	3-2
3.3	Nomeando valores	3-6
3.4	Avaliando expressões	3-7
3.5	Definindo variáveis e funções	3-8
3.6	Comentários	3-9
3.7	Definições locais em equações	3-10
3.8	Regra de <i>layout</i>	3-11
3.9	Comandos úteis do GHCi	3-12
3.10	Exercícios	3-12
3.11	Soluções	3-15
4	Tipos de Dados	4-1
4.1	Tipos	4-1
4.2	Alguns tipos básicos	4-1
4.3	Tipos função	4-3
4.4	Checação de tipos	4-3
4.5	Assinatura de tipo em definições	4-4
4.6	Consulta do tipo de uma expressão no GHCi	4-4
4.7	Soluções	4-6

5	Estruturas de dados básicas	5-1
5.1	Tuplas	5-1
5.2	Listas	5-2
5.2.1	Progressão aritmética	5-4
5.3	Strings	5-5
5.4	Valores opcionais	5-5
5.5	Exercícios	5-6
5.6	Soluções	5-7
6	Polimorfismo Paramétrico	6-1
6.1	Operação sobre vários tipos de dados	6-1
6.2	Variáveis de tipo	6-1
6.3	Valor polimórfico	6-2
6.4	Instanciação de variáveis de tipo	6-2
6.5	Algumas funções polimórficas predefinidas	6-2
6.6	Exercícios	6-3
6.7	Soluções	6-4
7	Sobrecarga	7-1
7.1	Sobrecarga	7-1
7.2	Algumas classes de tipo pré-definidas	7-2
7.2.1	Eq	7-2
7.2.2	Ord	7-3
7.2.3	Enum	7-3
7.2.4	Bounded	7-3
7.2.5	Show	7-4
7.2.6	Read	7-4
7.2.7	Num	7-4
7.2.8	Real	7-4
7.2.9	Integral	7-5
7.2.10	Fractional	7-5
7.2.11	Floating	7-5
7.2.12	RealFrac	7-6
7.2.13	RealFloat	7-6
7.3	Sobrecarga de literais	7-7
7.4	Conversão entre tipos numéricos	7-7
7.5	Inferência de tipos	7-8
7.6	Dicas e Sugestões	7-8
7.7	Exercícios	7-9
7.8	Soluções	7-11
8	Expressão Condicional	8-1
8.1	Expressão condicional	8-1
8.2	Definição de função com expressão condicional	8-2
8.3	Equações com guardas	8-3
8.4	Definições locais e guardas	8-5
8.5	Exercícios	8-7
8.6	Soluções	8-9

9	Programas Interativos	9-1
9.1	Interação com o <i>mundo</i>	9-1
9.1.1	Programas interativos	9-1
9.1.2	Linguagens puras	9-2
9.1.3	O mundo	9-2
9.1.4	Modificando o mundo	9-3
9.1.5	Ações de entrada e saída	9-3
9.2	O tipo <code>unit</code>	9-3
9.3	Ações de saída padrão	9-3
9.4	Ações de entrada padrão	9-4
9.5	Programa em Haskell	9-5
9.6	Combinando ações de entrada e saída	9-6
9.7	Exemplos de programas interativos	9-7
9.8	Saída bufferizada	9-9
9.9	Mais exemplos de programas interativos	9-11
9.10	Exercícios	9-12
9.11	Soluções	9-18
10	Funções Recursivas	10-1
10.1	Recursividade	10-1
10.2	Recursividade mútua	10-5
10.3	Recursividade de cauda	10-6
10.4	Vantagens da recursividade	10-9
10.5	Exercícios	10-9
10.6	Soluções	10-11
11	Ações de E/S Recursivas	11-1
11.1	A função <code>return</code>	11-1
11.2	Exemplo: exibir uma sequência	11-1
11.3	Exemplo: somar uma sequência	11-1
11.4	Problemas	11-3
11.5	Soluções	11-6
12	Casamento de Padrão	12-1
12.1	Casamento de padrão	12-1
12.1.1	Casamento de padrão	12-1
12.1.2	Padrão constante	12-2
12.1.3	Padrão variável	12-2
12.1.4	Padrão curinga	12-2
12.1.5	Padrão tupla	12-3
12.1.6	Padrões lista	12-3
12.1.7	Padrão lista na notação especial	12-4
12.2	Definição de função usando padrões	12-5
12.2.1	Definindo funções com casamento de padrão	12-5
12.3	Casamento de padrão em definições	12-9
12.4	Problema: validação de números de cartão de crédito	12-10
12.5	Problema: torres de Hanoi	12-12
12.6	Soluções	12-14

13	Expressão de Seleção Múltipla	13-1
13.1	Expressão case	13-1
13.2	Forma e regras de tipo da expressão case	13-1
13.3	Regra de <i>layout</i> para a expressão case	13-2
13.4	Avaliação de expressões case	13-2
13.5	Exemplos de expressões case	13-3
13.6	Expressão case com guardas	13-5
13.7	Soluções	13-7
14	Valores Aleatórios	14-1
14.1	Instalação do pacote random	14-1
14.2	Valores aleatórios	14-1
14.3	Jogo: adivinha o número	14-2
14.4	Soluções	14-9
15	Expressão Lambda	15-1
15.1	Valores de primeira classe	15-2
15.1.1	Valores de primeira classe	15-2
15.1.2	Valores de primeira classe: Literais	15-2
15.1.3	Valores de primeira classe: Variáveis	15-2
15.1.4	Valores de primeira classe: Argumentos	15-3
15.1.5	Valores de primeira classe: Resultado	15-3
15.1.6	Valores de primeira classe: Componentes	15-3
15.2	Expressão lambda	15-3
15.2.1	Expressões lambda	15-3
15.2.2	Exemplos de expressões lambda	15-4
15.2.3	Uso de expressões lambda	15-4
15.2.4	Exercícios	15-5
15.3	Aplicação parcial de funções	15-6
15.3.1	Aplicação parcial de funções	15-6
15.3.2	Aplicação parcial de funções: exemplos	15-6
15.4	<i>Currying</i>	15-8
15.4.1	Funções <i>curried</i>	15-8
15.4.2	Por que <i>currying</i> é útil?	15-8
15.4.3	Convenções sobre <i>currying</i>	15-8
15.5	Seções de operadores	15-9
15.5.1	Operadores	15-9
15.5.2	Seções de operadores	15-10
15.6	Utilidade de expressões lambda	15-11
15.6.1	Por que seções são úteis?	15-11
15.6.2	Utilidade de expressões lambda	15-12
15.6.3	Exercícios	15-14
15.7	Soluções	15-15
16	Funções de Ordem Superior	16-1
16.1	Funções de Ordem Superior	16-1
16.2	Um operador para aplicação de função	16-1
16.3	Composição de funções	16-2
16.4	A função filter	16-3
16.5	A função map	16-3
16.6	A função zipWith	16-4
16.7	As funções foldl e foldr , foldl1 e foldr1	16-4

16.7.1	<code>foldl</code>	16-4
16.7.2	<code>foldr</code>	16-5
16.7.3	<code>foldl1</code>	16-5
16.7.4	<code>foldr1</code>	16-6
16.8	<i>List comprehension</i>	16-6
16.8.1	<i>List comprehension</i>	16-6
16.8.2	<i>List comprehension</i> e funções de ordem superior	16-7
16.9	Cupom fiscal do supermercado	16-8
16.10	Soluções	16-13
17	Argumentos da Linha de Comando e Arquivos	17-1
17.1	Argumentos da linha de comando	17-1
17.2	Encerrando o programa explicitamente	17-2
17.3	Formatando dados com a função <code>printf</code>	17-4
17.4	Arquivos	17-5
17.5	As funções <code>lines</code> e <code>unlines</code> , e <code>words</code> e <code>unwords</code>	17-6
17.6	Exemplo: processar notas em arquivo	17-7
17.7	Problemas	17-8
17.8	Soluções	17-11
18	Tipos Algébricos	18-1
18.1	Novos tipos de dados	18-1
18.2	Tipos algébricos	18-2
18.3	Exemplo: formas geométricas	18-2
18.4	Exemplo: sentido de movimento	18-3
18.5	Exemplo: cor	18-5
18.6	Exemplo: coordenadas cartesianas	18-5
18.7	Exemplo: horário	18-5
18.8	Exemplo: booleanos	18-6
18.9	Exemplo: listas	18-6
18.10	Exercícios básicos	18-7
18.11	Números naturais	18-8
18.12	Árvores binárias	18-9
18.13	O construtor de tipo <code>Maybe</code>	18-9
18.14	Exercício: lógica proposicional	18-10
18.15	Soluções	18-16
19	Classes de Tipos	19-1
19.1	Polimorfismo <i>ad hoc</i> (sobrecarga)	19-1
19.2	Tipos qualificados	19-2
19.3	Classes e Instâncias	19-2
19.4	Tipo principal	19-3
19.5	Definição padrão	19-3
19.6	Exemplos de instâncias	19-4
19.7	Instâncias com restrições	19-4
19.8	Derivação de instâncias	19-5
19.8.1	Herança	19-5
19.9	Alguma classes do prelúdio	19-5
19.9.1	A classe <code>Show</code>	19-5
19.9.2	A classe <code>Eq</code>	19-6
19.9.3	A classe <code>Ord</code>	19-6
19.9.4	A classe <code>Enum</code>	19-6

19.9.5 A classe Num	19-7
19.10 Exercícios	19-8
19.11 Soluções	19-11
20 Valores em um Contexto	20-1
20.1 Valores encapsulados	20-1
20.2 Aplicação de função	20-2
20.3 Funtores	20-2
20.4 Funtores aplicativos	20-3
20.5 Mônadas	20-3
20.6 Exemplo: expressões aritméticas	20-5
20.7 Exemplo: geração de histórico	20-7
21 Parsers	21-1
21.1 Parsers	21-1
21.2 Parsers como funções	21-1
21.3 Soluções	21-3

2 Ambiente de Desenvolvimento Haskell

Resumo

As atividades de programação serão desenvolvidas usando a linguagem funcional Haskell (<http://www.haskell.org/>).

Nesta aula o aluno irá se familiarizar com o ambiente de programação em Haskell através da avaliação de expressões no ambiente interativo, e edição e compilação de programas. Também ele irá aprender a fazer suas primeiras definições de função.

Sumário

2.1	Haskell	2-1
2.2	Instalação do ambiente de desenvolvimento	2-2
2.2.1	Instalação das ferramentas de desenvolvimento em Haskell	2-2
2.3	O ambiente interativo GHCi	2-2
2.4	Bibliotecas	2-6

2.1 Haskell

Haskell é uma linguagem de programação **funcional pura** avançada. É um produto de **código aberto** de mais de vinte anos de pesquisa de ponta que permite o desenvolvimento rápido de software **robusto**, **conciso** e **correto**. Com um bom suporte para a **integração com outras linguagens**, **concorrência e paralelismo integrados**, **depuradores**, **ricas bibliotecas**, e uma **comunidade ativa**, Haskell pode tornar mais fácil a produção de software flexível, de alta qualidade, e de fácil manutenção.

GHC (Glasgow Haskell Compiler) (<http://www.haskell.org/ghc/>) é a implementação de Haskell mais usada:

- GHC é um software livre (de código aberto) para a linguagem Haskell.
- Está disponível para diversas plataformas, incluindo Windows e diversas variedades de Unix (como Linux, Mac OS X e FreeBSD).
- GHC é a implementação de Haskell mais usada.
- GHC compreende:
 - um **compilador** de linha de comando (**ghc**) usado para compilar programas gerando código executável
 - um **ambiente interativo** (**GHCi**), que permite a **avaliação de expressões** de forma interativa, muito útil para testes durante o desenvolvimento.
- Tem suporte particularmente bom para **concorrência** e **paralelismo**.
- Tem capacidades de **otimização**, incluindo otimização entre módulos.
- Gera código rápido, principalmente para programas concorrentes.

2.2 Instalação do ambiente de desenvolvimento

Para o desenvolvimento de aplicações na linguagem Haskell precisa-se minimamente de um **compilador ou interpretador de Haskell**, e de um **editor de texto** para digitação do código fonte. **Bibliotecas** adicionais e **ambientes integrados de desenvolvimento** também podem ser úteis.

São recomendados:

- **GHC**, o compilador de Haskell mais usado atualmente, que oferece também o ambiente interativo GHCi.
- **cabal-install** (<https://www.haskell.org/cabal>), um sistema para construção e empacotamento de bibliotecas e programas em Haskell. Fornece a ferramenta de linha de comando `cabal` que simplifica o processo de gerenciamento de software Haskell, automatizando a busca, configuração, compilação e instalação de bibliotecas e programas Haskell.
- **Haskell language server** (<https://github.com/haskell/haskell-language-server>), um servidor de linguagem que permite a integração com editores e ambientes de desenvolvimento.
- **Visual Studio Code** (<https://code.visualstudio.com>), um editor de texto com facilidades para o desenvolvimento de programas.

Uma outra ferramenta alternativa ao `cabal-install` é o **stack** (<https://docs.haskellstack.org/>). Na disciplina daremos preferência ao `cabal-install`.

2.2.1 Instalação das ferramentas de desenvolvimento em Haskell

A maneira recomendada de instalação das ferramentas de desenvolvimento em Haskell é usando o **GHCup** (<https://www.haskell.org/ghcup>). Com ele é possível instalar:

- GHC
- cabal-install
- haskell-language-server
- stack

Para realizar a instalação desses programas, siga as instruções encontradas na página web do GHCup.

Instale também o editor Visual Studio Code. No editor, instale a extensão Haskell.

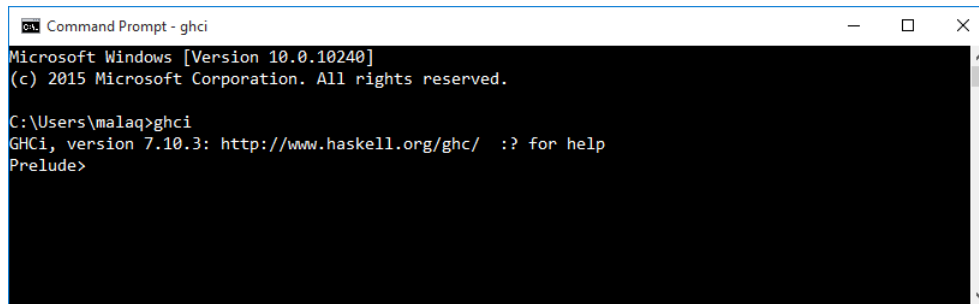
2.3 O ambiente interativo GHCi

O **GHCi** pode ser iniciado a partir de um terminal simplesmente digitando `ghci`. Isto é ilustrado na figura seguinte, em um sistema Unix.



```
Terminal
✓ [53912] 7:03:07 romildo jrm:~
% ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude>
```

No Windows pode-se iniciar o GHCi de maneira semelhante, a partir da janela *Prompt de Comandos*.



O prompt `Prelude>` significa que o sistema GHCi está pronto para avaliar expressões.

Uma aplicação Haskell é formada por um conjunto de módulos contendo definições de tipos, variáveis, funções, etc. À esquerda do prompt padrão do GHCi é mostrada a lista de módulos abertos (importados) que estão disponíveis. Um **módulo** é formado por definições que podem ser usadas em outros módulos. O módulo **Prelude** da biblioteca padrão do Haskell contém várias definições básicas e é importado automaticamente tanto no ambiente interativo quanto em outros módulos.

Na configuração padrão do GHCi o prompt é formado pela lista de módulos abertos seguida do símbolo `>`.

Expressões Haskell podem ser digitadas no prompt. Elas são compiladas e avaliadas, e o seu valor é exibido. Por exemplo:

```
Prelude> 2 + 3 * 4
14

Prelude> (2 + 3) * 4
20

Prelude> sqrt (3^2 + 4^2)
5.0
```

O GHCi também aceita **comandos** que permitem configurá-lo. Estes comandos começam com o caracter `:` (dois-pontos). Eles não fazem parte da linguagem Haskell. São específicos do ambiente interativo.

O comando `:quit` pode ser usado para encerrar a sessão interativa no GHCi. A sessão pode ser encerrada também pela inserção do caracter de fim de arquivo `Control-Z` no Windows e `Control--D` no Linux.

Normalmente a entrada para o GHCi deve ser feita em uma única linha. Assim que a tecla **ENTER** é digitada, encerra-se a leitura. Para realizar entrada usando várias linhas, pode-se delimitá-la pelos comandos `{` e `}`, colocados cada um em sua própria linha. Por exemplo:

```
Prelude> {:
Prelude| 2 + 3 * 4 ^
Prelude| 5 / (8 - 7)
Prelude| :}
3074.0
```

As linhas entre os delimitadores `{` e `}` são simplesmente unidas em uma única linha que será dada como entrada para o GHCi.

Alternativamente pode-se configurar o GHCi para usar o modo de linhas múltiplas por meio do comando `:set +m`. Neste modo o GHCi detecta automaticamente quando o comando não

foi finalizado e permite a digitação de linhas adicionais. Uma linha múltipla pode ser terminada com uma linha vazia. Por exemplo:

```
Prelude> :set +m

Prelude> sqrt (2 +
Prelude|   3 * 4)
3.7416573867739413
```

Pode-se obter ajuda no GHCi com os comandos :help ou :?.

```
Prelude> :help
Commands available from the prompt:

<statement>          evaluate/run <statement>
:                     repeat last command
:\n ..lines.. \n:\n   multiline command
:add [*]<module> ...   add module(s) to the current target set
:browse[!] [[*]<mod>]  display the names defined by module <mod>
                      (!: more details; *: all top-level names)
:cd <dir>             change directory to <dir>
:cmd <expr>           run the commands returned by <expr>::IO String
:complete <dom> [<rng>] <s> list completions for partial input string
:ctags[!] [<file>]    create tags file <file> for Vi (default: "tags")
                      (!: use regex instead of line number)
:def[!] <cmd> <expr>  define command :<cmd> (later defined command has
                      precedence, ::<cmd> is always a builtin command)
                      (!: redefine an existing command name)
:doc <name>           display docs for the given name (experimental)
:edit <file>          edit file
:edit                edit last module
:etags [<file>]       create tags file <file> for Emacs (default: "TAGS")
:help, :?            display this list of commands
:info[!] [<name> ...] display information about the given names
                      (!: do not filter instances)
:instances <type>    display the class instances available for <type>
:issafe [<mod>]       display safe haskell information of module <mod>
:kind[!] <type>       show the kind of <type>
                      (!: also print the normalised type)
:load[!] [*]<module> ... load module(s) and their dependents
                      (!: defer type errors)
:main [<arguments> ...] run the main function with the given arguments
:module [+/-] [*]<mod> ... set the context for expression evaluation
:quit               exit GHCi
:reload[!]          reload the current module set
                    (!: defer type errors)
:run function [<arguments> ...] run the function with the given arguments
:script <file>      run the script <file>
:type <expr>        show the type of <expr>
:type +d <expr>     show the type of <expr>, defaulting type variables
:unadd <module> ...  remove module(s) from the current target set
:undef <cmd>        undefine user-defined command :<cmd>
::<cmd>             run the builtin command
:!!<command>        run the shell command <command>

-- Commands for debugging:

:abandon            at a breakpoint, abandon current computation
:back [<n>]         go back in the history N steps (after :trace)
:break [<mod>] <l> [<col>] set a breakpoint at the specified location
:break <name>       set a breakpoint on the specified function
```

<code>:continue [<count>]</code>	resume after a breakpoint [and set break ignore count]
<code>:delete <number> ...</code>	delete the specified breakpoints
<code>:delete *</code>	delete all breakpoints
<code>:disable <number> ...</code>	disable the specified breakpoints
<code>:disable *</code>	disable all breakpoints
<code>:enable <number> ...</code>	enable the specified breakpoints
<code>:enable *</code>	enable all breakpoints
<code>:force <expr></code>	print <expr>, forcing unevaluated parts
<code>:forward [<n>]</code>	go forward in the history N step s(after :back)
<code>:history [<n>]</code>	after :trace, show the execution history
<code>:ignore <breaknum> <count></code>	for break <breaknum> set break ignore <count>
<code>:list</code>	show the source code around current breakpoint
<code>:list <identifier></code>	show the source code for <identifier>
<code>:list [<module>] <line></code>	show the source code around line number <line>
<code>:print [<name> ...]</code>	show a value without forcing its computation
<code>:sprint [<name> ...]</code>	simplified version of :print
<code>:step</code>	single-step after stopping at a breakpoint
<code>:step <expr></code>	single-step into <expr>
<code>:steplocal</code>	single-step within the current top-level binding
<code>:stepmodule</code>	single-step restricted to the current module
<code>:trace</code>	trace after stopping at a breakpoint
<code>:trace <expr></code>	evaluate <expr> with tracing on (see :history)

-- Commands for changing settings:

<code>:set <option> ...</code>	set options
<code>:seti <option> ...</code>	set options for interactive evaluation only
<code>:set local-config source ignore</code>	set whether to source .ghci in current dir (loading untrusted config is a security issue)
<code>:set args <arg> ...</code>	set the arguments returned by System.Environment.getArgs
<code>:set prog <progname></code>	set the value returned by System.Environment.getProgName
<code>:set prompt <prompt></code>	set the prompt used in GHCi
<code>:set prompt-cont <prompt></code>	set the continuation prompt used in GHCi
<code>:set prompt-function <expr></code>	set the function to handle the prompt
<code>:set prompt-cont-function <expr></code>	set the function to handle the continuation prompt
<code>:set editor <cmd></code>	set the command used for :edit
<code>:set stop [<n>] <cmd></code>	set the command to run when a breakpoint is hit
<code>:unset <option> ...</code>	unset options

Options for ':set' and ':unset':

<code>+m</code>	allow multiline commands
<code>+r</code>	revert top-level expressions after each evaluation
<code>+s</code>	print timing/memory stats after each evaluation
<code>+t</code>	print type after evaluation
<code>+c</code>	collect type/location info after loading modules
<code>-<flags></code>	most GHC command line flags can also be set here (eg. -v2, -XFlexibleInstances, etc.) for GHCi-specific flags, see User's Guide, Flag reference, Interactive-mode options

-- Commands for displaying information:

<code>:show bindings</code>	show the current bindings made at the prompt
<code>:show breaks</code>	show the active breakpoints
<code>:show context</code>	show the breakpoint context
<code>:show imports</code>	show the current imports
<code>:show linker</code>	show current linker state
<code>:show modules</code>	show the currently loaded modules
<code>:show packages</code>	show the currently active package flags

<code>:show paths</code>	show the currently active search paths
<code>:show language</code>	show the currently active language flags
<code>:show targets</code>	show the current set of targets
<code>:show <setting></code>	show value of <setting>, which is one of [args, prog, editor, stop]
<code>:showi language</code>	show language flags for interactive evaluation

The User's Guide has more information. An online copy can be found here:

https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/ghci.html

2.4 Bibliotecas

Os programas em Haskell são organizados em módulos. Um **módulo** é formado por um conjunto de **definições** (tipos, variáveis, funções, etc.). Para que as definições de um módulo possam ser usadas o módulo deve ser **importado**. Uma **biblioteca** é formada por uma coleção de módulos relacionados.

A **biblioteca padrão**¹ é formada por um conjunto de módulos disponível em todas as implementações de Haskell. Ela contém o módulo **Prelude**² que é *importado automaticamente por padrão em todos os programas em Haskell* e contém tipos e funções comumente usados.

A **biblioteca padrão do GHC**³ é uma versão expandida da biblioteca padrão contendo alguns módulos adicionais.

Hackage⁴ é uma coleção de **pacotes** contendo bibliotecas disponibilizados pela comunidade de desenvolvedores. Estes pacotes podem ser *instalados* separadamente.

Todas as definições de um módulo podem ser listadas no ambiente interativo usando o comando `:browse`. Exemplo:

```
Prelude> :browse Prelude
($!) :: (a -> b) -> a -> b
(!!) :: [a] -> Int -> a
($) :: (a -> b) -> a -> b
(&&) :: Bool -> Bool -> Bool
(++) :: [a] -> [a] -> [a]
(.) :: (b -> c) -> (a -> b) -> a -> c
(=<<) :: Monad m => (a -> m b) -> m a -> m b
data Bool = False | True
:
```

Tarefa 2.1

Use o ambiente interativo GHCi para avaliar todas as expressões usadas nos exemplos deste roteiro.

¹Veja <http://www.haskell.org/onlinereport/haskell2010/haskellpa2.html>.

²Veja <http://www.haskell.org/ghc/docs/latest/html/libraries/base/Prelude.html>.

³Veja <http://www.haskell.org/ghc/docs/latest/html/libraries/index.html>.

⁴Veja <http://hackage.haskell.org/>.

