

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	3-7
3.4	Avaliando expressões	3-8
3.5	Definindo variáveis e funções	3-8
3.6	Comentários	3-11
3.7	Definições locais em equações	3-11
3.8	Regra de <i>layout</i>	3-12
3.9	Comandos úteis do GHCi	3-13
3.10	Exercícios	3-13
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	Checagem 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

3 Expressões e Definições

Resumo

Neste capítulo são apresentados alguns elementos básicos da linguagem Haskell que permitirão a construção de expressões envolvendo constantes, variáveis e funções.

Sumário

3.1	Constantes	3-1
3.2	Aplicação de função	3-2
3.3	Nomeando valores	3-7
3.4	Avaliando expressões	3-8
3.5	Definindo variáveis e funções	3-8
3.6	Comentários	3-11
3.7	Definições locais em equações	3-11
3.8	Regra de <i>layout</i>	3-12
3.9	Comandos úteis do GHCi	3-13
3.10	Exercícios	3-13

3.1 Constantes

As formas mais básicas de expressões são os construtores constantes e os literais, que representam valores em sua forma mais simples, ou seja, já estão reduzidos à sua forma canônica. Uma expressão está na **forma canônica** quando ela não pode mais ser simplificada.

Os **literals** são expressões com sintaxe especial para escrever alguns valores. Já **construtores constantes** são identificadores começando com letra maiúscula.

Veja alguns exemplos de construtores constantes e literais na tabela a seguir.

descrição			exemplo
literais numéricos	inteiros	em decimal	8743
		em octal	0o7464
			00103
		em hexadecimal	0x5A0FF
	0xE0F2		
	fracionários	em decimal	140.58
			8.04e7
			0.347E+12
			5.47E-12
			47e22
literais caracter			'H'
			'\n'
			'\x65'
literais string			"bom dia"
			"ouro preto\nmg"
construtores booleanos			False
			True

Os literais numéricos são sempre positivos.

3.2 Aplicação de função

Aplicação de função é uma das formas de expressões mais comuns na programação funcional, uma vez que os programas são organizados em funções.

Sintaticamente uma aplicação de função em **notação prefixa** consiste em escrever a função seguida dos argumentos, se necessário *separados por caracteres brancos* (espaços, tabuladores, mudança de linha, etc.).

Exemplos:

```
Prelude> sqrt 25
5.0

Prelude> cos 0
1.0

Prelude> tan pi
-1.2246467991473532e-16

Prelude> exp 1
2.718281828459045

Prelude> logBase 3 81
4.0

Prelude> log 10
2.302585092994046
```

```
Prelude> mod 25 7
4

Prelude> negate 7.3E15
-7.3e15

Prelude> not True
False
```

Observe que, diferentemente de várias outras linguagens de programação, **os argumentos não são escritos entre parênteses e nem separados por vírgula.**

Parênteses podem ser usados para **agrupar subexpressões**. Por exemplo:

```
Prelude> sqrt (logBase 3 81)
2.0

Prelude> logBase (sqrt 9) 81
4.0
```

Aplicações de função também podem ser escritas em **notação infixa**, onde a função é escrita entre os seus argumentos. Neste caso dizemos que as funções são **operadores infixos**. Exemplos:

```
Prelude> 2 + 3
5

Prelude> 10 / 4
2.5

Prelude> (12 - 7) * 6
30

Prelude> 5 * sqrt 36
30.0

Prelude> 6 <= 17
True

Prelude> 'A' == 'B'
False

Prelude> 'A' /= 'B'
True

Prelude> True || False
True

Prelude> True && False
False
```

Assim como na Matemática e em outras linguagens de programação, os operadores possuem um **nível de precedência** (ou prioridade) e uma **associatividade**. Parênteses podem ser usados para agrupar subexpressões dentro de expressões maiores *quebrando* a precedência ou associatividade dos operadores.

O nível de precedência de um operador é dado por um número entre 0 e 9, inclusive. Se dois operadores disputam um operando, o operador de maior precedência é escolhido.

A tabela 3.1 lista os operadores definidos no prelúdio.

precedência	associatividade	operador	descrição
9	esquerda	!!	índice de lista
	direita	.	composição de funções
8	direita	^	potenciação com expoente inteiro não negativo
		^^	potenciação com expoente inteiro
		**	potenciação com expoente em ponto flutuante
7	esquerda	*	multiplicação
		/	divisão fracionária
		`div`	quociente inteiro truncado em direção a $-\infty$
		`mod`	módulo inteiro satisfazendo $(\text{div } x \ y) * y + (\text{mod } x \ y) == x$
		`quot`	quociente inteiro truncado em direção a 0
		`rem`	resto inteiro satisfazendo $(\text{quot } x \ y) * y + (\text{rem } x \ y) == x$
6	esquerda	+	adição
		-	subtração
5	direita	:	construção de lista não vazia
		++	concatenação de listas
4	não associativo	==	igualdade
		/=	desigualdade
		<	menor que
		<=	menor ou igual a
		>	maior que
		>=	maior ou igual a
		`elem`	pertinência de lista
		`notElem`	negação de pertinência de lista
3	direita	&&	conjunção (e lógico)
2	direita		disjunção (ou lógico)
1	esquerda	>>=	composição de ações sequenciais
		>>	composição de ações sequenciais (ignora o resultado da primeira)
0	direita	\$	aplicação de função
		\$!	aplicação de função estrita
		`seq`	avaliação estrita

Tabela 3.1: Precedências e associatividades dos operadores do **Prelude**.

Exemplos:

```
Prelude> 2 + 3 * 4      -- * tem maior precedência que +
14

Prelude> 5 ^ 2 - 10     -- ^ tem maior precedência que -
15

Prelude> 2 ^ 3 ^ 2      -- ^ associa-se à direita
512
```

Aplicações de função em notação prefixa tem prioridade maior do que todos os operadores.
Exemplos:

```
Prelude> abs 10 - 20      -- abs tem precedência maior que -
-10

Prelude> abs (10 - 20)
10

Prelude> succ 9 + max 5 4 * 3  -- succ e max tem precedência maior que + e *
25

Prelude> 2 * logBase (8/2) 256 + 1000
1008.0
```

Um operador pode ser associativo à esquerda, associativo à direita, ou não-associativo. Quando dois operadores com a mesma precedência disputam um operando,

- se eles forem associativos à esquerda, o operador da esquerda é escolhido,
- se eles forem associativos à direita, o operador da direita é escolhido,
- se eles forem não associativos, a expressão é mal formada e contém um erro de sintaxe,

Exemplos:

```
Prelude> 15 - 4 - 6      -- - associa-se à esquerda
5

Prelude> 15 - (4 - 6)
17

Prelude> 10 - 2 + 5      -- + e - tem a mesma precedência e associam-se à esquerda
13

Prelude> 10 - (2 + 5)
3

Prelude> 2^3^4           -- ^ associa-se à direita
2417851639229258349412352

Prelude> (2^3)^4
```

4096

```
Prelude> 3 > 4 > 5           -- > é não associativo
erro de sintaxe
```

O símbolo `-` merece atenção especial, pois ele pode tanto ser a função de subtração (operador infixo) como a função de inversão de sinal (operador prefixo).

```
Prelude> 6 - 2
4

Prelude> - 5
-5

Prelude> - (5 - 9)
4

Prelude> negate (5 - 9)
4

Prelude> 4 * (-3)
-12

Prelude> 4 * -3
erro de sintaxe
```

A notação prefixa é usada com nomes de funções que são identificadores alfanuméricos: formados por uma sequência de letras, dígitos decimais, sublinhado (`_`) e apóstrofo (`'`) começando com letra minúscula ou sublinhado (e que não seja uma palavra reservada).

Já a notação infixa é usada com nomes de funções simbólicos: formados por uma sequência de símbolos especiais (`! # $ % & + . / < = > ? @ | \ ^ - ~ :`) que não começa com `:`.

Qualquer operador pode ser usado em notação prefixa, bastando escrevê-lo entre parênteses. Exemplos:

```
Prelude> (+) 4 5
9

Prelude> (/) 18.2 2
9.1

Prelude> (>=) 10 20
False

Prelude> sqrt ((+) 4 5)
3
```

Qualquer função prefixa de dois argumentos pode ser usada em notação infixa, bastando escrevê-la entre apóstrofos invertidos (sinal de crase: ```), com precedência padrão 9 e associatividade à esquerda. Exemplos:

```
Prelude> 20 `div` 3
6
```

```
Prelude> 20 `mod` 3
2

Prelude> 20 `mod` 3 == 0
False

Prelude> 3 `logBase` 81
4.0

Prelude> (3 `logBase` 81) ^ 2
16.0

Prelude> 3 `logBase` (81 ^ 2)
8.0

Prelude> 3 `logBase` 81 ^ 2
16.0

Prelude> (20 `div` 3) ^ 2
36

Prelude> 20 `div` 3 ^ 2
2
```

3.3 Nomeando valores

Quando uma expressão é avaliada diretamente no ambiente interativo, uma variável chamada `it` é automaticamente definida para denotar o valor da expressão. Exemplo:

```
Prelude> 2 + 3 * 4
14

Prelude> it
14

Prelude> 7*(it - 4)
70

Prelude> it
70
```

Uma **declaração** `let` pode ser usada para definir uma variável no ambiente interativo. Por exemplo:

```
Prelude> let idade = 2 + 3 * 4

Prelude> idade
14
```

```
Prelude> 7*(idade - 4)
70
```

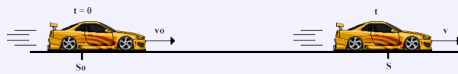
3.4 Avaliando expressões

Tarefa 3.1: Movimento Retilíneo Uniformemente Variado

A posição s de um corpo em movimento retilíneo uniformemente variado, em função do tempo t , é dado pela equação

$$s = s_0 + v_0 t + \frac{1}{2} a t^2$$

onde s_0 é a posição inicial do corpo, v_0 é a sua velocidade inicial, e a é a sua aceleração.



Utilize o **ambiente interativo GHCi** para calcular a posição de uma bola em queda livre no instante $t = 8$ s, considerando que a posição inicial é $s_0 = 100$ m, a velocidade inicial é $v_0 = 15$ m/s e a aceleração da gravidade é $a = -9.81$ m/s².

Dicas: Use a declaração `let` para criar variáveis correspondentes aos dados e em seguida avalie a expressão correspondente à função horária do movimento usando estas variáveis.

Tarefa 3.2: Expressões matemáticas

Utilize o **ambiente interativo** para avaliar as expressões aritméticas seguintes, considerando que $x = 3$ e $y = 4$.

a) $\frac{4}{3}\pi \sin x^2 - 1$

b) $\frac{x^2 y^3}{(x - y)^2}$

c) $\frac{1}{x^2 - y} - e^{-4x} + \sqrt[3]{\frac{35}{y}} \sqrt{xy}$

d) $\frac{24 + 4.5^3}{e^{4.4} - \log_{10} 12560}$

e) $\cos \frac{5\pi}{6} \sin^2 \frac{7\pi}{8} + \frac{\tan(\frac{\pi}{6} \ln 8)}{\sqrt{7} + 2}$

3.5 Definindo variáveis e funções

Além de poder usar as funções das bibliotecas, o programador também pode *definir* e *usar* suas próprias funções. Novas funções são definidas em arquivos texto geralmente chamados de **código fonte** ou **programa fonte** ou ainda **script**. Um programa fonte contém definições (de variáveis, funções, tipos, etc.) usadas para estruturar o código da aplicação.

Por convenção, *arquivos de programas fonte* em Haskell normalmente tem a *extensão* `.hs` em seu nome. Isso não é obrigatório, mas é útil para fins de identificação.

Variáveis e *funções* são definidas usando **equações**. No lado esquerdo de uma equação colocamos o nome da variável ou o nome da função seguido de seus parâmetros formais. No lado direito colocamos uma expressão cujo valor será o valor da variável ou o resultado da função quando a função for aplicada em seus argumentos.

Nomes de **funções** e **variáveis** podem ser alfanuméricos ou simbólicos:

identificadores alfanuméricos

- começam com uma letra minúscula ou sublinhado e podem conter letras, dígitos decimais, sublinhado (`_`) e apóstrofo (aspa simples `'`)
- são normalmente usados em notação prefixa
- exemplos:
`myFun`
`fun1`
`arg_2`
`x'`

identificadores simbólicos

- formados por uma sequência de símbolos e não podem começar com dois pontos (`:`)
- são normalmente usados em notação infixa
- exemplos:
`<+>`
`===`
`$*==$`
`+=`

Ao desenvolver um programa pode ser útil manter duas janelas abertas, uma executando um **editor de texto** para editar o código, e outra para o **ambiente interativo** (GHCi) em execução. Na seção 2.2 são apresentadas algumas sugestões de editores de texto que poderão ser usados para escrever seus programas.

Os arquivos de programas em Haskell sempre devem ser salvos usando a codificação de caracteres UTF-8.

Tarefa 3.3: Meu primeiro programa fonte

Inicie um editor de texto, digite as seguintes definições de função, e salve o arquivo com o nome `test.hs`.

```
-- calcula o dobro de um número
dobro x = x + x

-- calcula o quádruplo de um número
quádruplo x = dobro (dobro x)
```

Deixando o editor aberto, em outra janela execute o GHCi carregando o novo arquivo fonte:

```
$ ghci test.hs
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main                ( test.hs, interpreted )
Ok, modules loaded: Main.

*Main>
```

Agora, tanto `Prelude.hs` como `test.hs` são carregados, e as funções de ambos os arquivos fonte podem ser usadas:

```
*Main> quádruplo 10
40

*Main> 5*(dobro 2) - 3
17
```

Observe que o GHCi usa o nome de módulo `Main` se o arquivo fonte não define um nome para o módulo.

Tarefa 3.4: Modificando meu primeiro programa fonte

Deixando o GHCi aberto, volte para o editor, adicione as seguintes definições ao arquivo fonte `test.hs`, e salve-o.

```
areaCirculo r = pi * r^2
```

O GHCi não detecta automaticamente que o arquivo fonte foi alterado. Assim o comando **:reload** deve ser executado para que as novas definições possam ser usadas:

```
*Main> :reload
[1 of 1] Compiling Main                ( test.hs, interpreted )
Ok, modules loaded: Main.
```

```
*Main> areaCirculo 5
78.53981633974483
```

3.6 Comentários

Comentários são usados para fazer anotações no programa que podem ajudar a entender o funcionamento do mesmo. Os comentários são ignorados pelo compilador.

Um **Comentário de linha** é introduzido por `--` e se estende até o final da linha.

Um **Comentário de bloco** é delimitado por `{-` e `-}`. Comentários de bloco podem ser aninhados.

3.7 Definições locais em equações

Em Haskell **equações** são usadas para definir variáveis e funções, como discutido anteriormente. Em muitas situações é desejável poder definir valores e funções auxiliares em uma definição principal. Isto pode ser feito escrevendo-se uma **cláusula where** ao final da equação. Uma cláusula where é formada pela palavra chave **where** seguida das definições auxiliares.

A cláusula where faz **definições que são locais** à equação, ou seja, o **escopo** dos nomes definidos em uma cláusula where restringe-se à menor equação contendo a cláusula where.

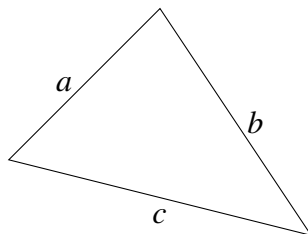
Por exemplo, considere a fórmula de Heron

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

para calcular a área de um triângulo com lados a , b e c , sendo

$$s = \frac{a + b + c}{2}$$

o semiperímetro do triângulo.



Como s aparece várias vezes na fórmula, podemos defini-lo localmente uma única vez e usá-lo quantas vezes forem necessárias na equação.

```
areaTriangulo a b c = sqrt (s * (s-a) * (s-b) * (s-c))
  where
    s = (a + b + c)/2
```

Esta definição assume que os argumentos da função são valores válidos para os lados de um triângulo.

```
areaTriangulo 5 6 8
~> sqrt (s * (s-5) * (s-6) * (s-8))
  where
    s = (5 + 6 + 8)/2
      ~> 9.5
~> sqrt (9.5 * (9.5-5) * (9.5-6) * (9.5-8))
~> sqrt 224.4375
~> 14.981238266578634
```

Tanto funções como variáveis podem ser definidas localmente. A ordem das equações locais é irrelevante. Por exemplo:

```
minhaFuncao x = 3 + f x + f a + f b
  where
    f x = x + 7*c
    a = 3*c
    b = f 2
    c = 10
```

```
minhaFuncao 5
~> 3 + f 5 + f a + f b
  where f x = x + 7*c
        a = 3*c
          ~> 3*10
          ~> 30
        b = f 2
          ~> 2 + 7*10
          ~> 2 + 70
          ~> 72
        c = 10
~> 3 + (5 + 7*10) + (30 + 7*10) + (72 + 7*10)
~> 3 + (5 + 70) + (30 + 70) + (72 + 70)
~> 3 + 75 + 100 + 142
~> 320
```

3.8 Regra de *layout*

Em uma **seqüência de definições**, cada definição deve começar precisamente na *mesma coluna*:

```
a = 10
b = 20
c = 30
```



```
a = 10
  b = 20
  c = 30
```



```
  a = 10
  b = 20
  c = 30
```



Se uma definição for escrita em mais de uma linha, as linhas subsequentes à primeira devem começar em uma coluna mais à direita da coluna que começa a seqüência de definições.

```
a = 10 + 20 + 30 +
    40 + 50 + 60 +
    70 + 80
b = sum [10,20,30]
```



```
a = 10 + 20 + 30 +
  40 + 50 + 60 +
  70 + 80
  b = sum [10,20,30]
```



```
a = 10 + 20 + 30 +
  40 + 50 + 60 +
  70 + 80
b = sum [10,20,30]
```



A **regra de layout** evita a necessidade de uma sintaxe explícita para indicar o agrupamento de definições usando `{`, `}` e `;`.

```
-- agrupamento implícito

a = b + c
  where
    b = 1
    c = 2

d = a * 2
```

significa

```
-- agrupamento explícito

a = b + c
  where { b = 1 ; c = 2 }

d = a * 2
```

Para evitar problemas com a regra de *layout*, é recomendado não *utilizar caracteres de tabulação para indentação do código fonte*, uma vez que um único caracterizar de tabulação pode ser apresentado na tela como vários espaços. O texto do programa vai aparentar estar alinhado na tela do computador, mas na verdade pode não estar devido ao uso do tabulador.

3.9 Comandos úteis do GHCi

comando	abrev	significado
:load <i>name</i>	:l	carrega o programa fonte <i>name</i>
:reload	:r	recarrega o programa fonte atual
:edit <i>name</i>	:e	edita o programa fonte <i>name</i>
:edit	:e	edita o programa fonte atual
:type <i>expr</i>	:t	mostra o tipo de <i>expr</i>
:info <i>name</i>	:i	dá informações sobre <i>name</i>
:browse <i>Name</i>		dá informações sobre o módulo <i>Name</i> , se ele estiver carregado
let <i>id</i> = <i>exp</i>		associa a variável <i>id</i> ao valor da expressão <i>exp</i>
:! <i>comando</i>		executa <i>comando</i> do sistema
:help	:h, :?	lista completa dos comandos do GHCi
:quit	:q	termina o GHCi

3.10 Exercícios

Nas tarefas seguintes, quando for solicitado para definir funções, elas devem ser definidas em um arquivo fonte e testadas no GHCi.

Tarefa 3.5: Encontrando os erros

Identifique e corrija os erros de sintaxe no programa fonte que se segue.

```
N = a 'div' length xs
  where
    a = 10
    xs = [1,2,3,4,5]
```

Tarefa 3.6

Defina uma função para calcular o quadrado do dobro do seu argumento.

Tarefa 3.7

Defina uma função para calcular o dobro do quadrado do seu argumento.

Tarefa 3.8: Lados de um triângulo

Os lados de qualquer triângulo respeitam a seguinte restrição:

A soma dos comprimentos de quaisquer dois lados de um triângulo é superior ao comprimento do terceiro lado.

Escreva uma função que receba o comprimento de três segmentos de reta e resulte em um valor lógico indicando se satisfazem esta restrição.

Tarefa 3.9: Energia armazenada em uma mola

A força requerida para comprimir uma mola linear é dada pela equação

$$F = kx$$

onde F é a força em N (Newton), x é a compressão da mola em m (metro), e k é a constante da mola em N/m .

A energia potencial armazenada na mola comprimida é dada pela equação

$$E = \frac{1}{2}kx^2$$

onde E é a energia em J (joule).

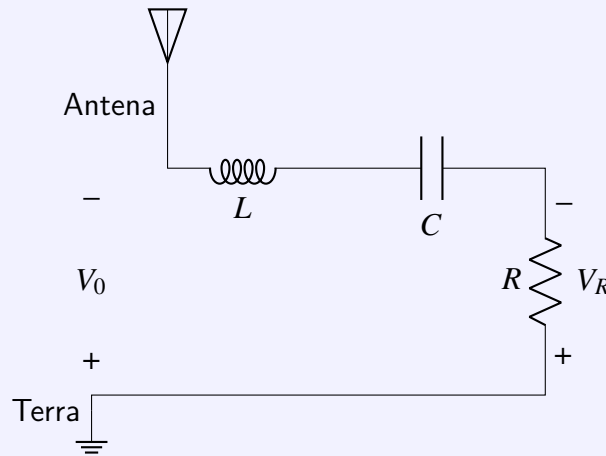
Defina funções para calcular a compressão e a energia potencial armazenada em uma mola, dadas a constante elástica da mola e a força usada para comprimi-la.

Tarefa 3.10: Custo da energia elétrica

Sabe-se que o quilowatt de energia elétrica custa um quinto do salário mínimo. Defina uma função que receba o valor do salário mínimo e a quantidade de quilowatts consumida por uma residência, e resulte no valor a ser pago com desconto de 15%.

Tarefa 3.11: Receptor de rádio

Uma versão simplificada da parte frontal de um receptor de rádio AM é apresentada na figura abaixo. Esse receptor é composto por um circuito que contém um resistor R , um capacitor C e um indutor L conectados em série. O circuito é conectado a uma antena externa e aterrado conforme mostra a figura.



O circuito permite que o rádio selecione uma estação específica dentre as que transmitem na faixa AM. Na frequência de ressonância do circuito, essencialmente todo o sinal V_0 da antena vai até o resistor, que representa o resto do rádio. Em outras palavras, o rádio recebe seu sinal mais forte na frequência de ressonância. A frequência de ressonância do circuito indutor-capacitor é dada pela equação

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

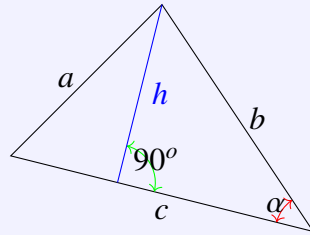
onde L é a indutância em H (henry) e C é a capacitância em F (farad).

Defina uma função que receba a indutância L e a capacitância C , e resulta na frequência de ressonância desse aparelho de rádio

Teste seu programa pelo cálculo da frequência do rádio quando $L = 0,25mH$ e $C = 0,10nF$.

Tarefa 3.12: Área de um triângulo usando relações métricas

A área de um triângulo de lados a , b e c pode ser calculada usando relações métricas em um triângulo qualquer.



Pela lei dos cossenos temos:

$$a^2 = b^2 + c^2 - 2bc \cos \alpha \implies \cos \alpha = \frac{b^2 + c^2 - a^2}{2bc}$$

Pela relação fundamental da trigonometria temos:

$$\sin^2 \alpha + \cos^2 \alpha = 1 \implies \sin \alpha = \sqrt{1 - \cos^2 \alpha}$$

Pela definição de seno temos:

$$\sin \alpha = \frac{h}{b} \implies h = b \sin \alpha$$

Pela definição da área de um triângulo temos:

$$A = \frac{ch}{2}$$

Defina uma função para calcular a área de um triângulo de lados a , b e c usando as equações apresentadas.

Dicas: Use variáveis locais para calcular os valores $\cos \alpha$, $\sin \alpha$ e h .

