

# DCC909 – Programação Funcional

**AULA 02**

**Carlos Bruno Oliveira Lopes**

*Engenheiro de Computação*

*Mestre em Ciência da Computação*

# Programação Funcional

## Haskell

- É uma linguagem de programação funcional;
  - Sua estrutura de controle primária é uma função.
- Empresas que utilizam Haskell:
  - **Bank of America Merrill Lynch** transformação de dados
  - **Facebook** manipulação da base de código PHP
  - **Google** infra-estrutura interna de TI
  - **NVIDIA** ferramentas usadas internamente
  - **The New York Times** processamento de imagens

# HASKELL

- GHC (*Glasgow Haskell Compiler*) é um compilador de código aberto para a linguagem Haskell;
  - Ele está disponível para diversas plataformas, tais como, Windows e diversas variedades de Unix (como Linux, Mac OS X e FreeBSD).
  - O compilador pode ser usado em linha de comando (ghc) ou integrado a um ambiente de interativo (GHCi);

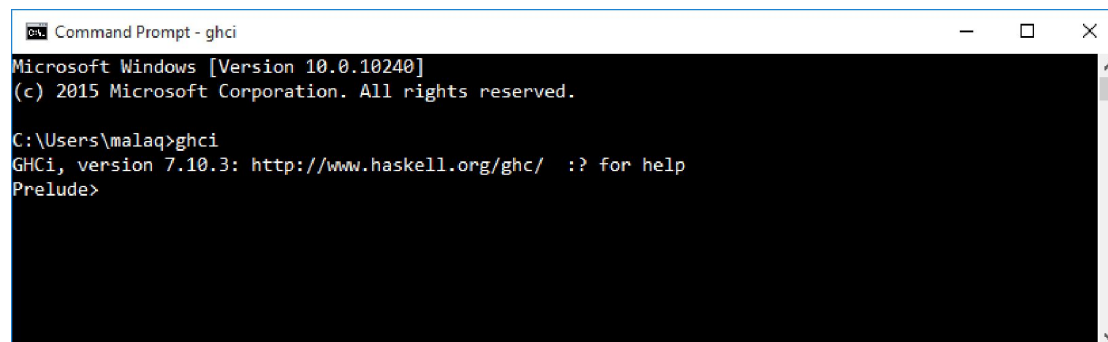
# HASKELL

- A Plataforma Haskell inclui:
  - o compilador GHC, incluindo o ambiente interativo GHCi,
  - WinGHCi, uma interface gráfica para o ambiente interativo do GHC,
  - cabal-install, e
  - bibliotecas adicionais.
- As instruções para instalação podem ser encontrados por acessando a pagina:
  - <https://www.haskell.org/>
  - Nessa pagina há instruções para instalação para Windows e linux

# HASKELL

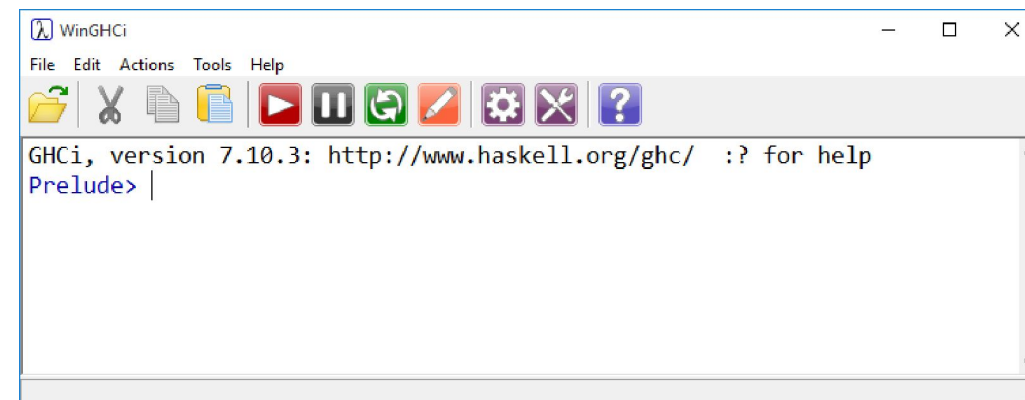
## Ambiente interativo GHCi

- O **GHCi** pode ser iniciado a partir de um terminal simplesmente digitando **ghci**.
- No Windows o programa **WinGHCi** é uma alternativa para executar o GHCi sem usar um terminal.



```
Command Prompt - ghci
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\malaq>ghci
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude>
```



```
WinGHCi
File Edit Actions Tools Help
GHCi, version 7.10.3: http://www.haskell.org/ghc/  :? for help
Prelude>
```

# HASKELL

- 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;
- **Biblioteca** é formada por uma coleção de módulos relacionados.
  - A biblioteca padrão é formada por um conjunto de módulos disponível em todas as implementações de Haskell;
  - **Prelude** é um módulo que é importado automaticamente por padrão em todos os programas em Haskell e contém tipos e funções comumente usados;
  - As bibliotecas da Plataforma Haskell são bibliotecas extras incluídas na plataforma Haskell;
  - **Hackage** é uma coleção de pacotes contendo bibliotecas disponibilizados pela comunidade de desenvolvedores.

# HASKELL

## Expressões

- As formas mais básicas de expressões no haskell são os construtores constantes e os literais;
  - **Literais** são expressões com sintaxe especial para escrever alguns valores.
  - **Construtores constantes** são identificadores começando com letra maiúscula.
    - Construtores são todos os valores que um tipo pode assumir.

# HASKELL

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



# HASKELL

## Aplicando Função

- As expressões lógicas ou soluções logicas em programação funcional são expressadas em formas de funções.
  - Todos os programas são organizados e desenvolvidos em funções.
- Na notação prefixada, as funções são expressadas pelo seu identificador (nome da função) seguido dos argumentos.
  - Ex.:

```
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
```

# HASKELL

## Aplicando Função

- Parênteses podem ser usados para agrupar subexpressões.
  - Ex.:

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

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

# HASKELL

## Aplicando Função

- Funções escrita na forma infixada, onde a função é escrita entre o seus argumentos.
- Chamamos essas funções de **operadores** infixos.

- Ex.:

```
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
```

# HASKELL

## Operadores

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

# HASKELL

## Operadores

– Ex.:

```
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
```

# HASKELL

## Operadores

– Ex.:

```
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
```

# HASKELL

## Operadores

— Ex.:

```
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
```

# HASKELL

## Operadores

### — Operador —

- Pode ser um função de subtração (operador infix)
- ou uma função de inversão de sinal (operador infix)
  - Ex.:

```
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
```



# HASKELL

## Operadores

- Qualquer operador pode ser usado em notação prefixada, no entanto ele deve ser usado entre parêntese.

- Ex.:

```
Prelude> (+) 4 5  
9
```

```
Prelude> (/) 18.2 2  
9.1
```

```
Prelude> (>=) 10 20  
False
```

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

# HASKELL

## Operador

- Qualquer função prefixa de dois argumentos pode ser usada em notação infixa, bastando escrevê-la entre apóstrofos invertidos (sinal de crase: `);

- Ex.:

```
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
```

# HASKELL

## Nomeando valores

- No ambiente interativo, há a variável **it** que armazena automaticamente o valor da última operação (ou função) executada.

- Ex.:

```
Prelude> 2 + 3 * 4
```

```
14
```

```
Prelude> it
```

```
14
```

```
Prelude> 7*(it - 4)
```

```
70
```

```
Prelude> it
```

```
70
```

# HASKELL

## Nomeando valores

- O comando **let** permite definir uma variável no ambiente interativo.
- Ex.:

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

```
Prelude> idade  
14
```

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

# HASKELL – exercício

## Questão 01: 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/s<sup>2</sup>.

**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.

# HASKELL – exercício

## Questão 02: 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)  $\cos \frac{5\pi}{6} \sin^2 \frac{7\pi}{8} + \frac{\tan(\frac{\pi}{6} \ln 8)}{\sqrt{7} + 2}$

# HASKELL

## Definindo variáveis e funções

- Funções podem ser definidas em arquivos de texto chamados códigos fonte ou programa fonte ou script.
  - Arquivos de programa fonte em Haskell são denotados pela extensão **.hs** no final do seu nome.
  - Variáveis e funções são definidas como “equações”.
    - Portanto, no lado esquerdo, colocamos o nome da variável ou nome da função seguido de seus parâmetros formais.
    - E 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.

# HASKELL

## Definindo variáveis e funções

- Nomes de funções 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
    - Ex.: `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
    - Ex:    `<+>`      `===`      `$*=*$`      `+=`



# HASKELL

## Primeiro programa

```
1  -- Definindo minha primeira funções em Haskell (Oi mundo)
2
3  main = putStrLn "Hello, World!"
4
5
6  a *** b = a^b  --redefinindo o simbolo de potência para **
7
8  areaDoQuadrado l = l^2
```

# HASKELL

## Comentários

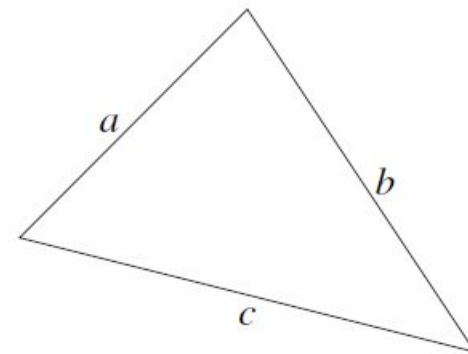
- **Comentário de linha** é introduzido por `--` e se estende até o final da linha.
- **Comentário de bloco** é delimitado por `{-` e `-}`.

# HASKELL

## Definições locais em equações

- A cláusula **where** permite a definição de equações locais (funções auxiliares em uma definição principal).
- Ex.: 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.

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



# HASKELL

## Definições locais em equações

– Tanto funções como variáveis podem ser definidas localmente.

• Ex.:

```
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
```

# HASKELL

## Regra de layout

*-- agrupamento implícito*

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

d = a * 2
```

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

significa

*-- agrupamento explícito*

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

d = a * 2
```

# HASKELL – Exercícios

1. Defina funções para o calcula a área de um círculo, retângulo, quadrado, trapézio e triângulo.
2. Defina funções para o calcula a área de um triângulo dados dois lados e um ângulo.
3. Defina funções para o calculo do perímetro de um retângulo e de círculo.
4. Defina uma função que calcula o volume de um cubo.
5. Defina uma função que calcula uma piscina de formato retangular.