Tipos de dados

- Tipos básicos
- Tipos função
- Checagem de tipos
- Assinatura de tipo em definições
- Consulta do tipo de uma expressão no GHCi

Tipos

- Um tipo é uma coleção de valores relacionados.
- Tipos servem para classificar os valores de acordo com as suas características.
- Em Haskell nomes de tipo são sequências de letras, dígitos decimais, sublinhados e apóstrofo, **começando com uma letra maiúscula.**
- Por exemplo, o tipo **Int** contém os valores inteiros, comumente usados nas **operações numéricas.**

Inteiros: Int (1)

- O tipo Haskell **Int** contém os inteiros.
- Os inteiros são os números inteiros, usados para contagem; eles são escritos assim:

45

- 8359

2147893748

Inteiros: Int (2)

- O tipo **Int** representa números inteiros em uma quantidade fixa de espaço

- Portanto, só pode representar um intervalo finito de números inteiros.

- O valor maxBound fornece o maior valor do tipo, que é 2147483647

Inteiros: Integer

- Para a maioria dos cálculos de inteiros, esses números de tamanho fixo são adequados,
- mas se números maiores forem necessários, podemos usar o tipo **Integer**,
- que pode representar com precisão números inteiros de qualquer tamanho.

Inteiros: operações

- Fazemos aritmética em inteiros usando os seguintes operadores e funções

+	**A soma de dois inteiros.**
*****	**O produto de dois inteiros.**
^	**Elevar a potência; 2^3 é 8.**
-	**A diferença de dois inteiros**
div	**Divisão de número inteiro; por exemplo, div 14 3 é 4.**
mod	**O restante da divisão de número inteiro**
abs	**O valor absoluto de um inteiro**
negate	**A função para alterar o sinal de um inteiro.**

Caracteres: Char

- Haskell contém um tipo embutido de caracteres, chamado **Char**.
- Os caracteres literais são escritos entre aspas simples, portanto, 'd' é o representante Haskell do caractere d.
- Alguns caracteres especiais são representados da seguinte forma:

tab	**'\t '**
newline	'\n'
backslash (\)	'\\'
single quote (')	'\''
double quote **(“)**	**'\”'**

Sequências de caracteres: String

- Haskell contém um tipo embutido de sequências de caracteres, chamado **String**.
- As sequências de caracteres são escritas entre aspas duplas,
- portanto, “funcional” é o representante Haskell da string funcional
- ‘c’ “ca sa”

Booleanos: Bool (1)

- O tipo booleano em Haskell é denominado **Bool**.
- O tipo **Bool** contém os dois valores lógicos False e True, comumente usados nas operações lógicas.

Booleanos: Bool (2)

- Os valores booleanos True e False representam os resultados de testes, que podem, por exemplo, comparar dois números para igualdade, ou verificar se o primeiro é menor que o segundo.

- Os operadores booleanos fornecidos na linguagem são:

&&	and	e
||	or	ou
not	not	não

Booleanos: Bool (3)

- Como Bool contém apenas dois valores, podemos definir o significado dos operadores booleanos por tabelas de verdade que mostram o resultado da aplicação do operador a cada combinação possível de argumentos.

F	F	F	F	T
F	T	F	T	T
T	F	F	T	F
T	T	T	T	F

Operadores relacionais

- Esses operadores recebem dois inteiros como entrada e retornam um Bool, que é True ou False. As relações são:

>	greater than (and not equal to)
>=	greater than or equal to
==	equal to
/=	Not equal to
<	less than or equal to
<=	less than (and not equal to)

Tipos básicos (1)

tipo	características	exemplos de valores
`Int`	– inteiros de precisão fixa – limitado (tem um valor mínimo e um valor máximo) – faixa de valores determinada pelo tamanho da palavra da plataforma	`876` `2012`
`Integer`	– inteiros de precisão arbitrária – ilimitado (qualquer número inteiro pode ser representado desde que haja memória suficiente) – menos eficiente que `Int`	`10` `754738748784003045452334209238`
`Float`	– aproximação de números reais em ponto flutuante – precisão simples	`4.56` `0.201E10`
`Double`	– aproximação de números reais em ponto flutuante – precisão dupla	`78643` `987.3201E-60`

`Rational`	– números racionais
	– precisão arbitrária
	– representados como uma razão de dois valores do tipo `Integer`
	– os valores podem ser construídos usando o operador `%` do módulo `Data.Ratio` (precedência 7 e associatividade à esquerda)
	`import Data.Ratio`
		`3 % 4`
		`8 % 2`
		`5 % (-10)`

`Bool`	– valores lógicos	`False` `True`

`Char`	– enumeração cujos valores representam caracteres unicode
	– estende o conjunto de caracteres ISO 8859-1 (latin-1), que é uma extensão do conjunto de caracteres ASCII
		`'B'`
		`'!'`
		`'\n'` nova linha
		`'\LF'` nova linha
		`'\^J'` nova linha
		`'\10'` nova linha
		`'\''` aspas simples
		`'\\'` barra invertida

`String`	– sequências de caracteres	`"Brasil"`
		`""`
		`"bom\ndia"`
		`"altura:\10\&199.4"`

Tipos básicos (2)

- Alguns literais são sobrecarregados.
- Isto significa que um mesmo literal pode ter mais de um tipo, dependendo do contexto em que é usado.
- O tipo correto do literal é escolhido pela análise desse contexto.
- Literais inteiros e literais fracionários —> Numéricos

Tipos básicos (3)

- Os literais inteiros podem ser de qualquer tipo numérico, como Int, Integer, Float, Double ou Rational, e

- Os literais fracionários podem ser de qualquer tipo numérico fracionário, como Float, Double ou Rational.

Tipos básicos (4)

- Por exemplo:
- O literal inteiro 2016 pode ser de qualquer tipo numérico (como Int, Integer, Float, Double ou Rational)
- O literal 5.61 pode ser de qualquer tipo fracionário (como Float, Double ou Rational).

Tipos função (1)

- Nas linguagens funcionais, uma função é um valor de primeira classe e,
- assim como os demais valores, tem um tipo.
- Este tipo é caracterizado pelos tipos dos argumentos e pelo tipo do resultado da função.

Tipos função (2)

- Em Haskell um tipo função é escrito usando o operador de tipo ->:
- t_1 -> ... t_{n-1}-> t_n
- onde
- $t_1,\ldots,t_{n-1}$ são os tipos dos argumentos
- t_n é o tipo do resultado

Tipos função (3)

- Exemplos:
- Bool -> Bool
  tipo das funções com um argumento do tipo Bool, e resultado do tipo Bool, como por exemplo a função not
- Bool -> Bool -> Bool
  tipo das funções com dois argumentos do tipo Bool, e resultado do tipo Bool, como por exemplo as funções (&&) e (||)
- Int -> Double -> Double -> Bool
  tipo das funções com três argumentos, sendo o primeiro do tipo Int e os demais do tipo Double, e o resultado do tipo Bool

Checagem de tipos (1)

- Toda expressão sintaticamente correta tem o seu tipo calculado em tempo de compilação.
- Se não for possível determinar o tipo de uma expressão ocorre um erro de tipo que é reportado pelo compilador.
- A aplicação de uma função a um ou mais argumentos de tipo inadequado constitui um erro de tipo.

Checagem de tipos (2)

- Por exemplo:
- Prelude> not ‘A’
- Couldn’t match expected type ‘Bool’ with actual type ‘Char’
- Neste exemplo o erro ocorre porque a função not, cujo tipo é Bool -> Bool, requer um valor booleano, porém foi aplicada ao argumento ’A’, que é um caracter.

Checagem de tipos (3)

- Haskell é uma linguagem fortemente tipada, com um sistema de tipos muito avançado.
- Todos os possíveis erros de tipo são encontrados em tempo de compilação (tipagem estática).
- Isto torna os programas mais seguros e mais rápidos, eliminando a necessidade de verificações de tipo em tempo de execução.

Assinatura de tipo em definições (1)

- Ao fazer uma definição de variável ou função, o seu tipo pode ser anotado usando uma assinatura de tipo imediatamente antes da equação.
- A anotação consiste em escrever o nome e o tipo separados pelo símbolo ::
- Exemplo de variável:
- notaFinal :: Double
- notaFinal = media 4.5 7.2

Assinatura de tipo em definições (2)

- exemplos de função:
- media :: Double -> Double -> Double
- media x y = (x + y)/2

- discriminante :: Double ->Double -> Double -> Double
- discriminante a b c = b^2 - 4*a*c

Assinatura de tipo em definições (3)

- Um exemplo simples usando essas definições é uma função para testar se três Ints são iguais.

```
threeEqual :: Int -> Int -> Int -> Bool

threeEqua1 m n p =  (m==n) && (n==p)
```

Assinatura de tipo em definições (4)

- Os booleanos podem ser os argumentos ou os resultados das funções.
- 'Exclusive or' é a função que retorna True exatamente quando um, mas não ambos os argumentos, tem o valor True.

```
exOr :: Bool -> Bool -> Bool

exOr x y = (x || y) && not (x && y)
```

Assinatura de tipo em definições (5)

- Os valores booleanos também podem ser comparados quanto à igualdade e desigualdade usando os operadores == e /=, que têm o tipo

  Bool -> Bool -> Bool

- Observe que /= é a mesma função que exOr, uma vez que ambos retornam o resultado True quando exatamente um de seus argumentos é True.

Consulta do tipo de uma expressão no GHCi

- No GHCi, o comando :type (ou de forma abreviada :t) calcula o tipo de uma expressão, sem avaliar a expressão.
- Exemplos:
- Prelude> :type not False

not False :: Bool

- Prelude> :type 'Z'

'Z' :: Char

Consulta do tipo de uma expressão no GHCi

- Mais exemplos:
- Prelude> :t 2*(5 - 8) <= 6 + 1

2*(5 - 8) <= 6 + 1 :: Bool

- Prelude> :type not

not :: Bool -> Bool

- Prelude> :t 69 -- 69 é de qualquer tipo p onde p é um tipo numérico

69 :: Num p => p

Exercício

- Crie o programa fonte aps3.hs (com todas as funções e variáveis) como codificado nesta aula.
- Teste todas as funções do programa fonte, carregando no GHCi.
- Observe a necessidade de definir os tipos de dados das variáveis e também das funções.