CURSO: Ciência da Computação PERÍODO: 4o. DISCIPLINA: Técnicas Alternativas de Programação	CÓDIGO: CI062 PROFESSOR: Andrey AULA: 20

1 Apresentação

Na aula de hoje vamos apresentar e discutir e exercitar os conceitos de tipos de dados em Haskell. A aula de hoje é baseada em [de Sá and da Silva, 2006]

2 Desenvolvimento

2.1 Tipos de Dados

Os principais tipos em Haskell são:

Tipo	Descrição	Exemplo
Bool	valores verdadeiro ou falso	True
Int	Números inteiros com restrição	938
Integer	Números inteiros sem restrição	2345678910234
Float	Números reais	.921
Char	Caracteres	'E'
String	Cadeias de caracteres	"palavra"

As definições de tipos das funções em Haskell seguem a seguinte definição

```
nome_da_funcao :: Tipo_arg1 -> Tipo_arg2 -> ... -> Tipo_argn -> Tipo_retorno
```

Exemplo:

```
d_AB :: Float -> Float -> Float -> Float -> Float
d_AB x1 y1 x2 y2 | x1 == x2 = abs (x2 - x1)
                 | y1 == y2 = abs (y2 - y1)
                 | otherwise = sqrt ((x2 - x1)^2 + (y2 - y1)^2)
```

O principais operadores matemáticos, de comparação e lógicos em Haskell são:
$+, -, *, /, \hat{}\ , ==, /=, <, >, >=, <=, ||, \&\&, not$

2.1.1 Operadores e Funções para os tipos de dados

O tipo booleano (Bool) possui dois valores True ou False. Além disso ele possui os seguintes operadores: '&&' (and), '||' (or) e 'not' (not) que servem para juntar dois ou mais valores lógicos de acordo com as condições da lógica proposicional comum.

O tipo Inteiro (Int) representa números inteiros e possui os seguintes operadores: $+, -, *$ e $\hat{}$ respectivamente para soma, subtração, multiplicação e exponenciação. Note que a divisão não está definida para tipos inteiros. A divisão entre números inteiros pode ser feita com a função **div** que retorna o quociente inteiro da divisão. Neste caso, é melhor usar o tipo Float. Além desses operadores, existem os operadores de comparação:

$==, /=, <, >, >=, <=$.

Para o tipo inteiro estão definidas as seguintes funções:

- 'div' que retorna o quocienta da divisão inteira entre dois números,
- 'mod' que retorna o resto da divisão inteira entre 2 números,
- 'abs' que retorna o valor absoluto do número e
- 'negate' que inverte o sinal do número.

O tipo Char representa um caractere (letra, número os sinal gráfico). Para o tipo Char estão definidas as seguintes funções:

- 'isLower' que retorna se a letra é minúscula,
- 'isUpper' que retorna se a letra é maiúscula,
- 'toLower' que converte a letra para minúscula,
- 'toUpper' que converte a letra para maiúscula,
- 'isDigit' que retorna se o caractere é um dígito,
- 'digitToInt' que converte o caractere (0 a 9) para a Inteiro,
- 'IntToDigit' que converte um Inteiro para um caractere (0 a 9).

Para representar os números reais, Haskell possui os tipos Float e Double. Os números Float são representados como números em ponto flutuante. O Double dobra a precisão de um número Float.

Além dos operadores do tipo Int, o tipo Float possui os operadores / para divisão e $**$ para a exponenciação onde os operandos podem ser números reais. Para o tipo Float estão definidas as seguintes funções:

- 'abs' que retorna o valor absoluto de um número,
- 'exp' que retorna o valor de e^x,
- 'fromIntegral' que converte de Int para Float,
- as funções trigonométricas 'acos', 'asin', 'atan', 'sin', 'cos' e 'tan',
- outras funções como 'log', logBase', 'negate', 'pi', 'signum', 'sqrt', 'ceiling', 'floor' e 'round'.

2.2 Tipos Sinônimos

Em Haskell é possível definir novos tipos com base nos tipos já existentes. Os tipos sinônimos são tipos que darão um outro nome a um tipo já existente ou então uma lista ou tupla envolvendo tipos já definidos. Esta definição é feita pelo construtor **type**. Por exemplo:

```
type Ponto = (Int, Int)
type Lista_Pontos = [Pontos]
```

A declaração type não permite a construção de tipos recursivos como os nós de uma árvore ou lista. O nome do tipo definido deve ter inicial maiúscula.

2.3 Tipos Estruturados

A criação de novos tipos é feita pelo pelo construtor **data**. Este tipo de construção permite a criação de tipos mais estruturados como enumerações e tipos com variações ou recursivos. Por exemplo:

```
data Cor = Verde | Azul | Amarelo
   deriving (Eq, Show)

corBasica :: Cor -> Bool
corBasica c = (c == Verde || c == Azul || c == Amarelo)
```

No exemplo, a expressão **deriving (Eq, Show)** significa que este novo tipo "herda"as propriedades das classes Eq e Show. A classe Eq permite a comparação de igualdade entre elementos deste tipo e a Show permite mostrar elementos deste tipo no terminal. Um outro exemplo é a definição dos tipos necessários para uma árvore.

```
data ArvoreBin = NodoNull | Nodo Int ArvoreBin ArvoreBin
      deriving Show
```

3 Atividades

1: Construa os tipos necessários para a implementação de uma agenda telefônica pessoal.

2: Implemente esta agenda telefônica com as funções de busca por nome e busca por número.

3: Construa os tipos para a implementação de uma lista encadeada.

4: Implemente a lista encadeada com as operações de inserir no começo, remover do começo e imprimir a lista.

Referências

[de Sá and da Silva, 2006] de Sá, C. C. and da Silva, M. F. (2006). *Haskell*. Novatec Editora.