CURSO: Ciência da Computação PERÍODO: 4o. DISCIPLINA: Técnicas Alternativas de Programação	CÓDIGO: CI062 PROFESSOR: Andrey AULA: 17

1 Apresentação

Na aula de hoje vamos apresentar e discutir e exercitar os conceitos de Programação em Haskell

2 Desenvolvimento

2.1 Haskell

É uma linguagem de programação funcional com tipos de dados bem definidos [Bird et al., 1998]. Consiste na escolha de funções no estilo matemático que expresse o problema em forma declarativa. Segundo [de Sá and da Silva, 2006], a linguagem Haskell possui as seguintes características:

- Estilo declarativo;
- Polimorfismo, Funções generalizadoras de alta ordem, avaliação de funções e expressões sob demanda (lazzy evaluation);
- Fácil e rápido aprendizado;
- Apresenta fundamentação teórica (Lambda Cálculo);
- Aplicações em prototipação, ensino de programação, Inteligência artificial, compiladores, ...

2.2 programando em Haskell

A programação em Haskell pode ser feita de forma compilada ou interativa. No início vamos tentar da forma iterativa usando o Glasgow Haskell Compiler (ghc). A programação funciona assim: As definições das funções são feitas em um arquivo texto com extensão .hs e a execução é feita diretamente na linha de comando. ex: vamos calcular a área de um triângulo com base x e altura y. No arquivo programas.hs colocamos a definição da função:

```
area_triangulo x y = x*y
```

No ambiente iterativo fazemos a carga do arquivo

```
prelude> :load programas.hs
```

A execução do programa:

```
prelude> area_retangulo 3 5
15
```

2.3 Funções em Blocos

Podemos usar funções para formar outras funções em Haskell, considerando a função como um bloco que executa uma operação. Exemplo, somando 2 números e usando esta função para somar 3 números:

```
soma x y = x+y

soma x y z = x + y + z

ou

soma x y z = soma (soma x y) z
```

2.4 Tipos de Parâmetros e Retorno

Em Haskel, podemos definir os tipos dos parâmetros e o tipo de retorno de uma função. Isto servirá como base para o compilador detectar uma possível conversão errada de tipos na chamada da função.

Na função que calcula a distancia entre dois números:

```
d_AB :: Int -> Int -> Int
d_AB x1 x2 = abs (x2 - x1)
```

Os dois primeiros Int são relativos aos parâmetros e o último é o tipo de retorno. 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"

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

2.5 Condições de Guarda

Quando precisamos tomar uma decisão sobre o valor a ser retornado pela função, podemos usar a notação de condição de guarda. Cada uma das condições de guarda são precedidas de '—', a seguir a condição e então o retorno. Exemplo, uma função que calcula a distância entre 2 pontos no plano:

```
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)
```

2.6 Recursividade

Em Haskell, como em todas as linguagens funcionalistas, a recursão é usada para se obter valores com cálculos iterativos. Ou seja, a recursão é usada no lugar de comandos iterativos como while ou for.

Por exemplo, se quisermos calcular a seguinte somatoria:

$\sum_{i=1}^{n} i$

Podemos usar a seguinte definição recursiva:

$$soma(x) = \begin{cases} 1 & x = 1 \\ x + soma(x-1) & x > 1 \end{cases}$$

O que dará a seguinte função em Haskell:

```
soma :: Int -> Int
soma 1 = 1
soma x = x + soma (x-1)
```

2.7 Problemas na Recursão

Em Haskell, as funções recursivas devem seguir uma sequência na sua definição. A definição da base (aterramento) deve vir antes da chamada recursiva (recursão). Isto se deve ao fato que Haskell seleciona as definições que irá adotar na ordem em que elas foram escritas no arquivo .hs. Por exemplo:

```
mult_7 3 = 0
mult_7 2 = 0
mult_7 5 = 0
mult_7 6 = 0
mult_7 4 = 0
mult_7 1 = 0
mult_7 7 = 0
mult_7 x = 1 + mult_7 (x-7)

Prelude> mult_7 9
1
Prelude> mult_7 18
2
```

Se invertermos duas, como no exemplo abaixo, sempre que for necessário calcular mult_7 de 2, o interpretador irá sempre entrar na linha de cima, empilhando chamadas recursivas até o estpuro da pilha.

```
mult_7 3 = 0
mult_7 5 = 0
mult_7 6 = 0
mult_7 4 = 0
```

```
mult_7 1 = 0
mult_7 7 = 0
mult_7 x = 1 + mult_7 (x-7)
mult_7 2 = 0

Prelude> mult_7 9

ERROR: Control stack overflow
```

2.8 Casamento de Padrões

Em Haskell é possível criar definições de funções que sejam ativadas e executadas sem considerar o conteúdo de um ou alguns dos argumentos da função. Isto é possível, usando-se a variável anônima '_'. *Porexemplo*:

```
g 7 y z = 10
g x 8 z = 20
g x y 9 = 30
g x y z | (x /= 7) || (y /= 8) || (z /= 9)  = 0
```

Pode ser reescrita desta forma:

```
g 7 _ _ = 10
g _ 8 _ = 20
g _ _ 9 = 30
g _ _ _ = 0
```

3 Atividades

1: Construa uma função que receba um número n e calcule o fatorial de n.

2: Construa uma função que receba três números a, b e c lados de um triãngulo e retorne 1 para equilátero, 2 para isósceles e 3 para escaleno.

3: Construa uma função que receba três números a, b e c e encontre o maior deles.

4: Construa uma função que receba três números a, b e c e retorne quantos estão acima do valor médio entre eles.

Referências

[Bird et al., 1998] Bird, R. et al. (1998). *Introduction to functional programming using Haskell*, volume 2. Prentice Hall Europe Hemel Hempstead, UK.

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