CURSO: Ciência da Computação PERÍODO: 4o. DISCIPLINA: Técnicas Alternativas de Programação	CÓDIGO: CI062 PROFESSOR: Andrey AULA: 19

1 Apresentação

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

2 Desenvolvimento

2.1 Tuplas

As tuplas em Haskell são coleções de dados heterogêneos de tamanho limitado. Servem para definir e manipular tipos de dados heterogêneos. Elas são baseadas em um conceito parecido com o das Structs em linguagem C.

Sua sintaxe básica é um conjunto de dados entre parênteses separados por vírgula. Por exemplo:

```
Prelude> (1,3)
(1,3)
Prelude> ("Joao", 10, 18.45, "Game")
("Joao", 10, 18.45, "Game")
```

Em Haskell podemos ter tuplas de diversos tamanhos e elas podem conter tipos simples e tipos definidos pelos programadores. Existem duas funções que manipulam tuplas, mas apenas para tuplas com 2 elementos. São elas: fst e snd. Por exemplo:

```
Prelude> fst (5,"Bom dia")
5
Prelude> snd (5,"Bom dia")
"Bom dia"
```

As tuplas poder ser usadas em funções como argumentos, retorno ou para manipulação dos elementos. Por exemplo:

```
soma_e_sub :: (Int, Int) -> (Int, Int)
soma_e_sub (a,b) = (a + b, a - b)
Prelude> soma_e_sub (1,2)
(3,-1)
```

2.2 Função Currificada

Currificação (Currying) é uma técnica de transformação de uma função que recebe múltiplos parâmetros (mais especificamente, uma n-tupla como parâmetro) de forma que ela pode ser chamada como uma cadeia de funções que recebem somente um parâmetro cada. Foi inventada por Moses Schönfinkel e Gottlob Frege, e independentemente por Haskell Curry. Por exemplo:

```
soma_e_sub_currificada :: Int -> Int -> (Int, Int)
soma_e_sub_currificada a b = (a + b, a - b)
```

Uma função de n parãmetros pode ser resumida em uma função de um único parâmetro. O modo como ocorrem os passos abaixo só é possível pelo fato dos parãmetros estarem livres entre si.

```
f_currificada x y z w = x + y + z + w
Prelude> f_currificada 3 4 5 6
18
Prelude> (f_currificada 3) 4 5 6
18
Prelude> ((f_currificada 3) 4) 5 6
18
Prelude> (((f_currificada 3) 4) 5) 6
18
Prelude> ((((f_currificada 3) 4) 5) 6)
18
```

2.3 Tipos compostos por Tuplas

Novos tipos em Haskell podem ser compostos pela função type. São conhecidos como tipos algébricos. Seu funcionamento é análogo ao do typedef do C. Por exemplo:

```
type Peso = Float
type Idade = Int
type Pessoa = (String, Idade, Peso, String)
f_Joao :: Pessoa
f_Joao = ("Joao", 19, 75.78, "Basquete")
```

Podemos criar algumas funções para selecionar campos das Tuplas.

```
selec_nome :: Pessoa -> String
selec_idade :: Pessoa -> Idade
selec_peso :: Pessoa -> Peso
selec_esporte :: Pessoa -> String

selec_nome (n,i,p,e) = n
selec_idade (n,i,p,e) = i
selec_peso (n,i,p,e) = p
selec_esporte (n,i,p,e) = e
```

2.4 Tuplas como bases de dados

Vamos dar um exemplo de como podemos usar as tuplas para manipular dados. Primeiro vamos declarar o tipo referente à tupla que iremos usar com os dados que queremos manipulas. Os dados são: nome, idade e gênero.

```
type Meu_tipo = (String, Float, Char)
```

Agora vamos declarar uma função que, de acordo com o valor do RG, retorna a tupla com os dados requeridos.

```
pessoa :: Float -> Meu_tipo
pessoa rg
        |rg == 1 = ("joao", 12, 'm')
        |rg == 2 = ("jonas", 81, 'm')
        |rg == 3 = ("joice", 21, 'f')
        |rg == 4 = ("janete", 55, 'f')
        |rg == 5 = ("jocileide", 21, 'f')
        |otherwise = ("ninguem", 0, 'x')
```

Agora podemos criar funções para manipular as tuplas com os dados. Primeiro vamos encontrar a menor idade da relação. Para isso, vamos criar uma função que encontra a menor idade entre duas tuplas, uma função que retorna a idade e outra que retorna o gênero de uma tupla. Após vamos criar uma função que encontra a menor idade entre as tuplas retornadas pela função pessoa, percorrendo todas as tuplas pela variação do rg.

```
idade (n1, i1, g1) = i1
genero (n1, i1, g1) = g1

menor (n1, i1, g1) (n2, i2, g2)
        | i1 <= i2 = (n1, i1, g1)
        | otherwise = (n2, i2, g2)

menor_idade :: Float -> Meu_tipo
menor_idade x
        | x == 1 = pessoa 1
        | otherwise = menor (pessoa x) (menor_idade (x-1))
```

Agora vamos criar uma função para calcular a média das idades dos registros. Esta função depende de uma função que percorre todos as tuplas e soma as idades. Uma outra função interessante é a função que conta quantas pessoas são de um determinado gênero. Ela chama uma função que percorre as tuplas e compara o gênero passado com o gênero da tupla e soma 1 se for o mesmo.

```
media_idade :: Float -> Float
media_idade x = (soma_idade x)/x

soma_idade x
        |x == 1 = idade (pessoa 1)
        |otherwise = idade (pessoa x) + (soma_idade (x-1))

conta_genero :: Float -> Char -> Float
```

```
conta_genero x g = conta x g 0

conta :: Float -> Char -> Float -> Float
conta x g cont
        |x == 0 = cont
        |genero (pessoa x) /= g = conta (x-1) g cont
        |otherwise = conta (x-1) g (cont + 1)
```

3 Atividades

1: Construa uma função que receba um caractere qualquer e retorno uma tupla com o caractere minúsculo, o caractere maiúsculo número e o seu código ASCII.

2: Construa uma função que receba quatro números a, b, c e d e retorne uma tupla com os quatro números ordenados.

3: Construa uma função que, dada a base abaixo, retorne a) o número de doutores; b) o número de mulheres doutoras.

```
base :: Float -> Meu_tipo
base x
        |x == 1 = ("joao", "mestre" 'm')
        |x == 2 = ("jonas", "doutor", 'm')
        |x == 3 = ("joice", "mestre", 'f')
        |x == 4 = ("janete", "doutor", 'f')
        |x == 5 = ("jocileide", "doutor", 'f')
        |otherwise = ("ninguem", "", 'x')
```

Referências

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