Programação Funcional

(COMP0393)

Leila M. A. Silva

dcomp
departamento de computação

Módulos

(COMP0393)

Aula 15

dcomp
departamento de computação

Módulos

- Arquivo que define algumas funções, tipos e classes de tipos.
- Um programa em Haskell é uma coleção de módulos.
- Um módulo pode **exportar** as suas definições para serem usadas por outros módulos.
- Importante em programas de grande porte porque permite reuso de partes de programas e também melhor manutenibilidade.
- Haskell inclui vários módulos, cada um servindo a um propósito comum. Ex: módulo para manipular listas, módulo para manipular números complexos, etc...
- O Prelude é um módulo da biblioteca de Haskell que é importado por *default*.

Importação de Módulos

- Sintaxe:

```
import NomeDoModulo
```

- O módulo `Data.List` inclui um conjunto de funções para manipular listas. Ex:

```
import Data.List

numUnicos :: (Eq a) => [a] -> Int
numUnicos = length.nub
```

Função que dada uma lista calcula o número de elementos distintos

Função que dada uma lista elimina elementos que são repetidos

Bibliotecas em Haskell

- https://hoogle.haskell.org/
- https://downloads.haskell.org/~ghc/latest/docs/html/libraries/
- No GHCi, você pode usar:

```
ghci> :m + Data.List

ghci> :m + Data.List Data.Map Data.Set
```

Bibliotecas em Haskell

- No arquivo .hs você usa

```
import Data.List

import Data.List (nub, sort)


import Data.List hiding (nub)


import qualified Data.Map

Data.Map.filter

import qualified Data.Map as M

M.filter
```

Importa todo o módulo

Importa somente as funções listadas

Importa tudo exceto as funções listadas

Importa tudo mas exige que o uso de funções do módulo venha sempre precedido do nome do módulo para evitar colisão de nomes

Leitura Obrigatória

- Acesse as bibliotecas de Haskell nos links indicados e descubra as várias funções que elas oferecem, especialmente `Data.List`, `Data.Map` e `Data.Char`.

Exercícios Recomendados

- Usando funções da bibloteca `Data.List` elabore uma função que conta quantas vezes uma palavra ocorre dentro de um texto. Para isto faça:
  - Transforme um texto em uma lista de palavras
  - Ordene palavras de uma lista
  - Agrupe palavras que são adjacentes idênticas numa lista, construindo uma lista de listas de palavras idênticas
  - Finalize construindo uma função que dado uma lista de listas de palavras idênticas devolve uma lista de tuplas (palavra, quantidade).

Exercícios Recomendados

- Usando funções da bibloteca `Data.List` elabore :
  - Função que identifica se uma palavra dada é prefixo de alguma outra em uma lista de palavras.
  - Função que identifica se uma lista está totalmente contida em outra lista (todos os elementos da lista contida aparecem consecutivamente na lista original).
- Usando funções da bibloteca `Data.Char` elabore :
  - Função para realizar a cifração de César. A cifração de César transforma uma mensagem em outra deslocando cada letra do alfabeto de um número fixo. Ex: Palavra original: cama; Palavra cifrada supondo deslocamento de 3: fdpd.

Criando seus Módulos

- Considere a criação de um módulo que provê algumas funções para calcular o volume e a área de algumas formas geométricas.
- Defina o nome do módulo e este será o nome do seu arquivo. Ex: `Geometria.hs`

Criando seus Módulos

```
module Geometria (volEsfera, areaEsfera,
                  volCubo, areaCubo, volCuboide,
                  areaCuboide) where
volEsfera :: Float -> Float
volEsfera raio = (4.0/3.0) * pi * (raio ^ 3)

areaEsfera :: Float -> Float
areaEsfera raio = 4 * pi * (raio ^2)

volCubo :: Float -> Float
volCubo lado = volCuboide lado lado lado

areaCubo :: Float -> Float
areaCubo lado = areaCuboide lado lado lado

volCuboide :: Float -> Float -> Float -> Float
volCuboide a b c = retArea a b * c

areaCuboide :: Float ->Float -> Float -> Float
areaCuboide a b c = retArea a b * 2 + retArea a c * 2 + retArea c b * 2

retArea :: Float -> Float -> Float
retArea a b = a*b
```

Funções que são exportadas

Não é exportada, modificações nesta função não serão percebidas pelos usuários do módulo

Módulos Hierárquicos

- Considere dividir o módulo Geometria em três submódulos

Arquivo `Esfera.hs`

```
module Geometria.Esfera (volume, area) where

volume :: Float -> Float
volume raio = (4.0/3.0) * pi * (raio ^ 3)

area :: Float -> Float
area raio = 4 * pi * (raio ^2)
```

Arquivo `Cuboide.hs`

```
module Geometria.Cuboide (volume, area) where

volume :: Float -> Float -> Float -> Float
volume a b c = retArea a b * c

area :: Float ->Float -> Float -> Float
area a b c = retArea a b * 2 + retArea a c * 2 + retArea c b * 2

retArea :: Float -> Float -> Float
retArea a b = a*b
```

Não é exportada

Módulos Hierárquicos

Arquivo `Cubo.hs`

```
module Geometria.Cubo (volume, area) where

import qualified Geometria.Cuboide as Cuboide

volume :: Float -> Float
volume lado = Cuboide.volume lado lado lado

area :: Float -> Float
area lado = Cuboide.area lado lado lado
```

Importa de `Cuboide`

Diretório `Geometria` deve conter os três módulos

```
import Geometria.Esfera  -- para usar apenas um submódulo

-- para usar os três submódulos sem conflito de nome
import qualified Geometria.Esfera as Esfera
import qualified Geometria.Cuboide as Cuboide
import qualified Geometria.Cubo as Cubo

-- usa as funções como
Esfera.area, Cubo.area e Cuboide.area, etc...
```

Exercícios Recomendados

- Estenda o exemplo anterior criando submódulos para outras formas geométricas, como cilindro, cone, tronco de cone, etc...
- Crie um módulo que contenha algoritmos de ordenação já vistos e além das funções que ordenam incorpora funções para checar se uma lista está ordenada em ordem crescente ou decrescente.