Módulos em Haskell

INE 5416/5636 - Paradigmas de programação

Turmas 04208/08238

Prof. Dr. João Dovicchi – dovicchi@inf.ufsc.br

http://www.inf.ufsc.br/˜dovicchi

Módulos

Em HASKELL os módulos contém declarações de funções, classes, instâncias, tipos etc. que podem ser utilizadas por outros programas.

Módulos

Em HASKELL os módulos contém declarações de funções, classes, instâncias, tipos etc. que podem ser utilizadas por outros programas.

Vantagens como em C:

- Facilitam a manutenção do código
- Permitem a reutilização de código

Módulos em Haskell

Conteúdo dos Módulos:

- declarações de precedência (fixity)
- declarações de tipos
- declarações de classes e instâncias
- definições de tipos
- definições de funções

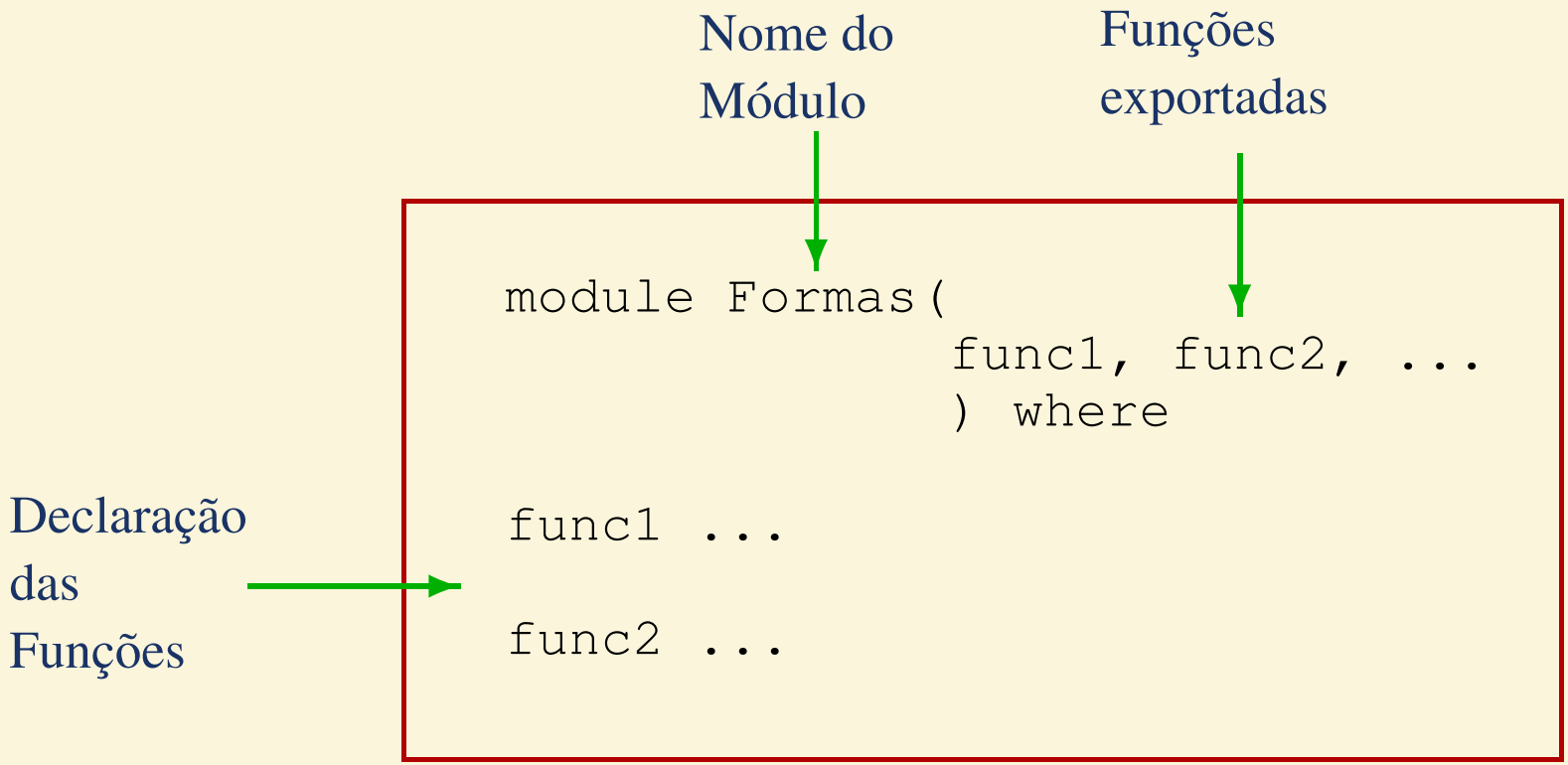
Módulos

O nome do módulo deve ser a primeira declaração, podendo ser precedida apenas por pragmas do compilador, quando for o caso.

Nome do módulo deve ser alfa-numérico e iniciar com maiúsculas.

Nome do módulo e nome do arquivo não têm, necessariamente, que ter o mesmo formato ou nome.

Exemplo de Módulo



Compilação

mod1.hs

mod2.hs

mod3.hs $\Longleftrightarrow$ main.hs

...

modN.hs

Um exemplo

```
module Fatorial (fact) where

fact 0 = 1
fact n = n * fact (n-1)
```

```
module Main (main) where

import Fatorial

main = print (fact 20)
```

ghc -o fat fatorial.hs main.hs

Componentes: `data`

A palavra "`data`" é usada para tipos definidos pelo usuário em Haskell.

```
data Cores = vermelho
           | verde
           | azul
           | branco
           | amarelo
           | preto
```

Módulos em Haskell

Exemplo:

```
data Forma = Retangulo Float Float
           | Elipse Float Float
           | TriangRet Float Float
           | Poligono [(Float,Float)]
     deriving Show
```

Obs.: lembre-se que a classe `Show` converte tipos para ser exibidos no formato de caracteres.

Componentes: `type`

A palavra "`type`" é usada para sinônimos de tipos em Haskell.

```
type String     = [Char]
type Person     = (Name,Address)
type Name       = String
type Address    = String
```

Módulos em Haskell

Exemplo:

```
data Forma = Retangulo Lado Lado
           | Elipse Raio Raio
           | TriangRet Lado Lado
           | Poligono [Vertice]
     deriving Show

type Raio    = Float
type Lado    = Float
type Vertice = ( Float , Float )
```

Componentes: declarações

Pode-se, em HASKELL declarar dados a partir de outros:

```
quadrado s    = retangulo s s
circulo r     = elipse r r
```

Componentes: funções

Funções são declaradas dentro de módulos

```
area :: Forma -> Float
area ( Retangulo s1 s2 ) = s1 * s2
area ( TriangRet s1 s2 ) = s1 * s2 /2
area ( Elipse r1 r2 )    = pi * r1 * r2
```

Importação

Módulos podem importar outros módulos. Exemplo:

```
module Mod1 (size, double, square) where

size :: Int
size = 12+13

double :: Int -> Int
double n = 2*n

square :: Int -> Int
square n = n*n
```

```
module Mod2 where

import Mod1

ex1, ex2 :: Int
ex1 = double 32 - square (size - double 3)
ex2 = double 320 - square (size - double 6)
```

Importação

Pode-se importar apenas parte de um módulo. Ex.:

```
odule Main where
 import System (readLn)

 main :: IO ()
 main = do
        putStrLn "Entre uma lista de inteiros: "
        args <- readLn
        let result = maxList args
        putStrLn . unwords $ [show result] /+$+/

 maxList [] = 0
 maxList (x:xs) = maxi x (maxList xs)
    where
       maxi a b | a <= b    = b
                | otherwise = a
```

Importação

Pode-se importar um módulo, excluindo-se funções que podem ser, então, redefinidas.

```
module Local (max, toUpper, isDigit) where
import Prelude hiding (max,toUpper,isDigit)

max :: Int -> Int -> Int
max x y
   | x >= y       = x
   | otherwise    = y

toUpper :: Char -> Char
toUpper ch = chr (ord ch + offset)

isDigit :: Char -> Bool
isDigit ch = ('0' <= ch) && (ch <= '9')
```

Note que, `Max, toUpper` e `isDigit` são definidos no Prelude, neste caso, importa-se o Prelude menos as funções dentro da cláusula 'hiding'.