Linguagem de Programação
Introdução à Linguagem Hugs98

TURMA A
PROF. MARCELO LADEIRA CIC/UNB

Funções e Operadores

São estruturas da LF que aplicadas a parâmetros e operandos retornam valores.

Funções são em geral prefixadas, associativas à esquerda e com prioridade máxima (10).

- Exemplo: f a b = (f a) b

Operadores são em geral infixados e podem ter suas prioridades e associatividades declaradas.

- Exemplo: x^y^2 = x^(y^2), mas
  x+y+2 = (x+y)+2

Funções

Tem nomes com letras e dígitos, começando com letra. São prefixadas:

a) fatorial n = product [1..n]

b) mdc a b
| b == 0 = a
| otherwise = mdc b (a `mod` b)

UMA FUNÇÃO PODE SER TRATADA DE FORMA INFIXADA SE FOR ENVOLVIDA POR APÓSTROFES.

Função Lambda

Função anônima. É definida como:

(\args -> corpo)

Exemplo: (\x y -> x^2 + y^2)

```
Prelude> (\x -> x^2) 3
9 :: Integer
Prelude> map (\x ->x^2) [1,2,3,4]
[1,4,9,16] :: [Integer]
```

Funções com Guardas

Uma função é definida como:

eq_1	mdc a 0 = a
eq_2	mdc a b = mdc b (mod a b)
...	fat 0 = 1
eq_n	fat (n+1) = (n+1) * fat n

Onde cada equação tem uma das formas:

a) f p_1 p_2 ... p_k | <guarda> = expressão

b) f p_1 p_2 ... p_k = expressão

```
mdc a b
    | b == 0   = a
    | otherwise = mdc b (mod a b)
```

Expressões Case

Uma expressão *case* tem a forma geral:

case <exp> of { p_1 match$_1$; ... ; p_n match$_n$ }

```
merge xs ys = case (xs,ys) of
        (z:zs,w:ws) | z<=w      -> z:merge zs ys
                    | z>w       -> w:merge xs  ws
        ([],ws)                 -> ys
        (zs,[])                 -> xs
```

```
Main> merge [1,3..9] [2,4..10]
[1,2,3,4,5,6,7,8,9,10] :: [Integer]
```

Expressões Case

```
fat n = case n of
            0       -> 1
            1       -> 1
           (k+1)    -> (k+1)*fat k
```

if e_1 then e_2 else e_3
= case e_1 of { True -> e_2 ; False -> e_3 }

e2 e e3 podem ser if também!

Expressões Case

```
qsort ls =
   case ls of
                  []          -> []
                  [x]         -> [x]
                  otherwise -> qsort ys ++ [x] ++ qsort zs
            where
                (x:xs) = ls
                ys = [y | y <- xs, y <  x]
                zs = [z | z <- xs, z >= x]
Main> qsort [8,5,7,3,4]
   [3,4,5,7,8] :: [Integer]
```

Expressões Case

```
pega n ys = case (n,ys) of
        (0,_)     ->  []
        (_,[])    ->  []
        (n,x:xs) ->  x : pega (n-1)  xs
```

```
Main> pega 3 [8,5,7,3,4]
    [8,5,7] :: [Integer]
Main> pega 7 [8,5,7,3,4]
    [8,5,7,3,4] :: [Integer]
```

Operadores

Operador tem nome formado por símbolo especial (não letras ou dígito) e é infixado:

```
Prelude> map (\x ->x^2) [1,2,3,4]
[1,4,9,16] :: [Integer]
Prelude> 5 == 9
False :: Bool
Prelude> [1,2,3] ++ [5,6]          ou          (++) [1,2,3] [5,6]
[1,2,3,5,6] :: [Integer]
```

Operadores

Operadores são definidos de forma similar a funções:

```
[] \/ ys = ys
(x:xs) \/ ys | membro x ys  = xs \/ ys
             | otherwise    = x: xs \/ ys
membro z [] = False
membro z (w:ws) = z==w || membro z ws
```

Funções e operadores podem fazer uso de definições locais com let ou where

```
xs \/ ys      | xs == [] = ys
              |  membro x ys = resto
              | otherwise = x:resto where (x:t)=xs;resto = t \/ ys
```

Operadores

A definição de um operador leva em conta:

- prioridade
- associatividade
- comportamento

Prioridade (inteiro entre 1 e 9)

2*3+4 = (2*3) +4 =10 ?

= 2*(3 + 4) =14 ?

Associatividade

1 - 2 - 3 = (1-2)-3 = -4 ?

= 1-(2-3) = 2

x⊕y⊕z = (x ⊕ y) ⊕ z - à esquerda infixl

= x ⊕ (y ⊕ z) - à direita infixr

= erro! - não associado. infix