Expressões lambda

- Valores de primeira classe
- Expressão lambda
- Aplicação parcial de funções

Valores de primeira classe

- Tipo de primeira classe: não há restrições sobre como os seus valores podem ser usados.

- São valores de primeira classe:
  - números – caracteres – tuplas – listas
  - funções
  - entre outros

Valores de primeira classe: literais

- Valores de vários tipos podem ser escritos literalmente, sem a necessidade de dar um nome a eles:

valor	tipo	descrição
`True`	`Bool`	o valor lógico *verdadeiro*
`'G'`	`Char`	o caracter *G*
`456`	`Num a => a`	o número 456
`2.45`	`Fractional a => a`	o número em ponto flutuante 2.45
`"haskell"`	`String`	a cadeia de caracteres *haskell*
`[1,6,4,5]`	`Num a => [a]`	a lista dos números 1, 6, 4, 5
`("Ana",False)`	`([Char],Bool)`	o par formado por *Ana* e *falso*

Valores de primeira classe: variáveis

- Valores de vários tipos podem ser nomeados:

```
matricula   = 456
sexo        = 'M'
aluno       = ("Ailton Mizuki Sato",101408,'M',"com")
disciplinas = ["BCC222","BCC221","MTM153","PRO300"]
livroTexto  = ("Programming in Haskell","G. Hutton",2007)
```

Valores de primeira classe: argumentos

- Valores de vários tipos podem ser argumentos de funções:

```
sqrt 2.45
not True
length [1,6,4,5]
take 5 [1,8,6,10,23,0,0,100]
```

Valores de primeira classe: resultado

- Valores de vários tipos podem ser resultados de funções:

```
not False        ⇝ True
length [1,6,4,5] ⇝ 4
snd ("Ana",'F')  ⇝ 'F'
tail [1,6,4,5]   ⇝ [6,4,5]
```

Valores de primeira classe: componentes

- Valores de vários tipos podem ser componentes de outros valores (estruturas de dados):

```
("Ana",'F',18)
["BCC222","BCC221","MTM153","PRO300"]
[("Ailton",101408),("Lidiane",102408)]
```

Valores de primeira classe: funções

- Funções também podem ser escritas sem a necessidade de receber um nome:

valor	**tipo**	**descrição**
`\x -> 3*x`	`Num a => a -> a`	função que calcula o triplo
`\n -> mod n 2 == 0`	`Integral a => a -> Bool`	função que verifica se é par
`\(p,q) -> p+q`	`Num a => (a,a) -> a`	função que soma par

Valores de primeira classe: funções

- Funções também podem ser nomeadas:
- triplo = \x -> 3 * x
- Esta equação define a variável triplo, associando-a a um valor que é uma função.
- Haskell permite escrever esta definição de forma mais sucinta (visto anteriormente):
- triplo x = 3 * x

Valores de primeira classe: funções

- Funções também podem ser argumentos de outras funções:
- map triplo [1,2,3]
- ==> [3,6,9]
- A função triplo é aplicada a cada elemento da lista [1,2,3], resultando na lista [3,6,9]

Valores de primeira classe: funções

- Funções também podem ser resultados de outras funções:
- (abs . sin) (3*pi/2)
- ==> 1.0
- (sqrt . abs) (-9)
- ==> 3.0
- O operador binário infixo (.) faz a composição de duas funções.

Valores de primeira classe: funções

- Funções também podem ser componentes de outros valores (estruturas de dados):

- map (\g -> g (-pi)) [abs, sin, cos]
- ==> [3.141592653589793, -1.2246467991473532e-16, -1.0]

- O segundo argumento de map é a lista das funções abs, sin e cos.

Expressões lambda

- Da mesma maneira que um número inteiro, uma string ou um par podem ser escritos sem ser nomeados,
- Uma função também pode ser escrita sem associá-la a um nome.
- As chamadas **funções anônimas**.
- λ

Expressões lambda

- Expressão lambda é uma função anônima (sem nome),
  - representada pelo caracter \
- formada por uma seqüência de padrões representando os argumentos da função,
- e um corpo que especifica como o resultado pode ser calculado usando os argumentos:
- \padrão$_1$ . . . padrão$_n$ -> expressão
- \x -> 3 * x

Expressões lambda

- O termo lambda provém do cálculo lambda
- Introduzido por Alonzo Church nos anos 1930 como parte de uma investigação sobre os fundamentos da Matemática.
- O cálculo lambda é uma teoria de funções na qual as linguagens funcionais se baseiam

Expressões lambda

- No cálculo lambda expressões lambdas são introduzidas usando a letra grega λ.
- $\lambda x.x+x$
- Em Haskell usa-se o caracter \, que se assemelha um pouco com λ.
- \x -> x + x

exemplos de expressões lambda

- Função anônima que calcula o dobro de um número:
- \x -> x + x
- O tipo desta expressão lambda é Num a => a -> a

- Função anônima que mapeia um número x a 2x + 1:
- \x -> 2*x + 1
- cujo tipo é Num a => a -> a

exemplos de expressões lambda

- Função anônima que calcula o fatorial de um número:
- \n -> product [1..n]
- cujo tipo é (Enum a, Num a) => a -> a

- Função anônima que recebe três argumentos e calcula a sua soma:
- \a b c -> a + b + c
- cujo tipo é Num a => a -> a -> a -> a

exemplos de expressões lambda

- Definições de função usando expressão lambda:
- f = \x -> 2*x + 1
- somaPar = \(x,y) -> x + y
- fatorial = \n -> product [1..n]
- é o mesmo que (visto anteriormente):
- f x = 2*x+1
- somaPar (x,y) = x + y
- fatorial n = product [1..n]

uso de expressões lambda

- Apesar de não terem um nome, funções construídas usando expressões lambda podem ser usadas da mesma maneira que outras funções.
- Como fazer as aplicações de função usando expressões lambda ?
- Isto é, como passar os argumentos para uma função anônima ?
- \x -> 2*x + 1

uso de expressões lambda

- Exemplos de aplicações de função usando expressões lambda:
- (\x -> 2*x + 1) 8
- ==>17
- (\x y -> sqrt (x*x + y*y)) 3 4
- ==>5.0

uso de expressões lambda

- Exemplos de aplicações de função usando expressões lambda:
- (\a -> (a, 2*a, 3*a)) 5
- ==> (5,10,15)
- (\(x1,y1) (x2,y2) -> sqrt ((x2-x1)^2 + (y2-y1)^2)) (6,7) (9,11)
- ==> 5.0

Aplicação parcial de funções

- Uma função com múltiplos argumentos pode também ser considerada como uma função que retorna outra função como resultado.
- qualquer função que receba dois ou mais argumentos pode ser parcialmente aplicada a um ou mais argumentos.
- Isso fornece uma maneira poderosa de formar funções como resultados.

Aplicação parcial de funções

- Exemplo de aplicação parcial:
- multiplica :: Int -> Int -> Int
- multiplica x y = x*y
- essa função recebe dois argumentos
- mas é possível fazer somente x = 2
- a função como resultado será 2 * y

Aplicação parcial de funções: exemplos

- Seja a seguinte função:
- f :: Int -> Int -> Int
- f x y = 2*x + y
- A função f recebe dois argumentos inteiros x e y e resulta na soma 2*x + y.

- f 2 3
- ==>7

Aplicação parcial de funções: exemplos

- Alternativamente esta função pode ser definida em duas etapas:
- f' :: Int -> (Int -> Int)
- f' x = h
  where h y = 2*x + y
- A função f' recebe um argumento inteiro x e resulta na função h, que por sua vez recebe um argumento inteiro y e calcula 2*x + y.

Aplicação parcial de funções: exemplos

- Aplicando a função:
- f’ 2 3
- ==>(f’ 2) 3
- ==>h 3
- ==>2*2 + 3
- ==>7
- As funções f e f’ produzem o mesmo resultado final, mas f foi definida de uma forma mais breve.

Aplicação parcial de funções: exemplos

- Podemos ainda definir a função usando uma expressão lambda:
- f’’ :: Int -> (Int -> Int)
- f’’ x =
  \y -> 2*x + y
- Da mesma forma que f’, a função f’’ recebe um argumento inteiro x e resulta em uma função.
- Esta função recebe um argumento inteiro y e calcula 2*x + y.

Aplicação parcial de funções: exemplos

- Aplicando a função:
- f’’ 2 3
- ==>(f’’ 2) 3
- ==>(\y -> 2*2 + y) 3
- ==>2*2 + 3
- ==>7

Aplicação parcial de funções: exemplos

- Podemos ainda definir a função usando duas expressões lambda:
- f’’’ :: Int -> (Int -> Int)
- f’’’ =
  \x -> (\y -> 2*x + y)

Aplicação parcial de funções: exemplos

- Aplicando a função:
- f’’’ 2 3
- ==>(\x -> (\y -> 2*x + y)) 2 3
- ==>(\y -> 2*2 + y) 3
- ==>2*2 + 3
- ==>7

Aplicação parcial de funções: exemplos

- Todas as versões apresentadas para a função f (f, f’, f’’ e f’’’) são equivalentes.
- Portanto a função f pode ser considerada como uma função que recebe um argumento e resulta em outra função que, por sua vez, recebe outro argumento e resulta na soma do dobro do primeiro argumento com o segundo argumento.

Aplicação parcial de funções: exemplos

- Isto permite a aplicação parcial da função:
- map (f 2) [1,8,0,19,5]
- ==> [5,12,4,23,9]
- (f 3 . length) "entendeu?"
- ==>15
- filter (not . even . f 10) [1,8,0,19,5]
- ==>[1,19,5]

Aplicação parcial de funções: exemplos

- Outro exemplo: multiplicação de três números:
- mult :: Int -> Int -> Int -> Int
- mult x y z = x * y * z
- A função mult recebe três argumentos e resulta no produto destes argumentos.
- Na verdade mult recebe um argumento de cada vez.

Aplicação parcial de funções: exemplos

- Ou seja, mult recebe um inteiro x e resulta em uma função que por sua vez recebe um inteiro y e resulta em outra função, que finalmente recebe um inteiro z e resulta no produto x * y * z.
- A aplicação de função tem associatividade à esquerda.
- mult x y z
- significa
- ((mult x) y) z

Aplicação parcial de funções: exemplos

- Este entendimento fica claro quando usamos expressões lambda para definir a função de maneira alternativa:
- mult’ :: Int -> (Int -> (Int -> Int))
- mult’ =

  \x-> \y-> \z -> x*y*z