Linguagens de Programação

Fabio Mascarenhas - 2013.1

http://www.dcc.ufrj.br/~fabiom/lp

Expressões condicionais

- Scala tem uma expressão `if-else` para expressar escolha entre alternativas que se parece muito com a estrutura de controle de Java, mas é usado com expressões ao invés de comandos (e é uma expressão, ou seja, avalia para um valor)

- `def abs(x: Int) = if (x >= 0) x else -x`

- A condição de uma expressão `if-else` deve ter tipo Boolean

Expressões booleanas

- Expressões booleanas podem ser
  - Constantes `true` e `false`
  - Negação: `!b`
  - Conjunção (e): `a && b`
  - Disjunção (ou): `a || b`
  - Operadores relacionais: `e1 <= e2`, `e1 == e2`, `e1 != e2`, `e1 >= e2`, `e1 < e2`, `e1 > e2`

Avaliação de expressões booleanas

- A avaliação se expressões booleanas segue as seguintes *regras de reescrita* (a expressão do lado esquerdo é substituída pela do lado direito, onde *e* é uma expressão qualquer):
  - `!true --> false`
  - `!false --> true`
  - `true && e --> e`
  - `false && e --> false`
  - `true || e --> true`
  - `false || e --> e`
- Note que && e || são operadores de “curto-circuito”, ou seja, às vezes eles não precisam avaliar ambos os operandos

Avaliação do if-else

- As regras de avaliação de uma expressão `if-else` são intuitivas:
  - `if(true) e1 else e2 --> e1`
  - `if(false) e1 else e2 --> e2`
- Naturalmente, primeiro é preciso avaliar a expressão condicional até se obter seu valor booleano!

val vs. def

- Até agora usamos `def` para definir tanto valores quanto funções, mas para valores o normal em Scala é usar val
  - `val raio = 10`
- A diferença entre `def` e `val` para valores é a mesma entre parâmetros CBN e CBV, com `val` vamos sempre avaliar o lado direito da definição, e o valor resultante é usado
- Fica óbvio se o lado direito da definição é uma expressão que não termina!

Exemplo: raiz quadrada

- Vamos definir uma função para calcular a raiz quadrada de um número, usando o método de Newton (aproximações sucessivas)

- `def raiz(x: Double) = ...`

- Começamos com uma *estimativa* y para a raiz de x (por ex., y = 1), obtemos a média entre y e x/y para ter uma nova estimativa, e repetimos o processo até o grau de precisão desejável

- Exemplo para x = 2

Implementação Raiz Quadrada

- `def raizIter(est: Double, x: Double): Double = if (suficiente(est, x)) est else raizIter(melhora(est, x), x)`
  - Função recursiva que computa um passo do processo
- `def suficiente(est: Double, x: Double) = abs(quadrado(est) - x) < 0.001`
  - Já temos precisão suficiente
- `def melhora(est: Double, x: Double) = (est + x / est) / 2`
  - Melhora a estimativa

Blocos

- As funções auxiliares que fazem parte da implementação de `raiz` (`raizIter`, `suficiente`, `melhora`) não precisam ficar visíveis para o programa todo
- Podemos defini-las dentro de `raiz` usando um *bloco* como corpo de `raiz`
- Um bloco é delimitado por `{ }`, e é uma expressão que contém uma sequência de definições e expressões
- O *último* elemento do bloco deve ser uma expressão que vai dar o valor de todo o bloco
- As definições em um bloco só são visíveis dentro desse bloco

Exercício: blocos e escopo

- Qual o valor de `result` no programa abaixo?

```
val x = 0
def f(y: Int) = y + 1
val result = {
  val x = f(3)
  x * x
}
```

Recursão Final

- Sejam as duas funções abaixo
  - `def mdc(a: Int, b: Int): Int = if (b == 0) a else mdc(b, a % b)`
  - `def fat(x: Int): Int = if (x < 2) 1 else x * fat(x - 1)`
- Vamos avaliar `mdc(14, 21)` e `fat(4)` passo a passo
- Qual a diferença entre as duas sequências?

Recursão Final

- Se o tamanho do termo sendo avaliado permanece em uma faixa constante durante o processo de avaliação, então deve ser possível implementar o processo de avaliação em uma quantidade constante de memória!
  - A recursão em `mdc` (e em `raizIter`) não precisa "estourar a pilha"
  - Esse tipo de chamada de função tem o nome de *recursão final* (*tail recursion*), ou *chamada final* (*tail call*)
- Geralmente linguagens funcionais implementam chamadas finais dessa forma, mas Scala, por limitações da JVM, não faz isso por padrão

Recursão Final em *Scala*

- Se uma função recursiva usa recursão final, você pode anotar sua definição com a anotação @tailrec, e o compilador Scala vai otimizar a chamada recursiva

- Se a chamada não for final o compilador vai reclamar

```
@tailrec
def mdc(a: Int, b: Int): Int =
  if (b == 0) y else mdc(b, a % b)
```