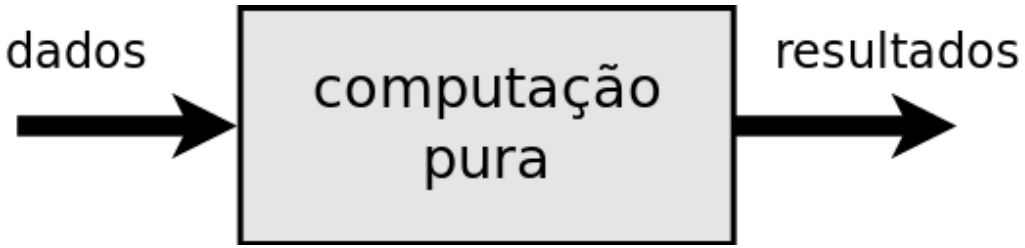
Programação Funcional
Aula 9 — Programas interativos

Pedro Vasconcelos
DCC/FCUP

2021

Motivação

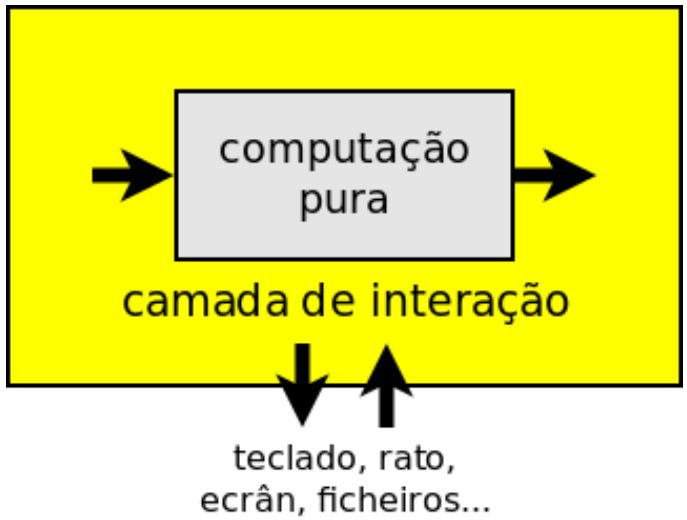
Até agora apenas escrevemos funções puras, i.e. funções que recebem todos os dados necessário antes de iniciar a computação.



Motivação (cont.)

Vamos agora ver como escrever programas interativos i.e. que durante a computação interagem com o mundo:

- lêndo informação do teclado ou de ficheiros;
- escrevendo no terminal ou em ficheiros.



Problema

- As funções em Haskell são sempre puras — ou seja, só podem depender dos valores do seus argumentos
- Isto obriga a usar um mecanismo especial para fazer programas interativos — que são necessariamente impuros
- A solução adotada recorre a um tipo especial

  ```
  IO a
  ```

  para as ações que podem fazer interação e depois devolvem um resultado do tipo `a`

Ações básicas

```
getChar :: IO Char          -- ler um carater

putChar :: Char -> IO ()    -- imprimir um carater

return :: a -> IO a
    -- retornar um valor sem fazer qualquer interação
```

Se introduzir uma ação IO no GHCi, esta é executada:

Exemplo

```
> putChar 'A'
A>
```

Encadear ações

Podemos combinar ações de I/O usando notação-do.

```
> do putChar 'A'; putChar 'B'; putChar 'C'
ABC>
```

Num programa podemos definir ações usando notação-do com indentação:

```
ação = do putChar 'A'
          putChar 'B'
          putChar 'C'
```

Encadear ações (cont.)

Podemos usar `<-` para obter o valor retornado por uma ação.

Exemplo

Ler um caracter e imprimi-lo duas vezes.

```
ação = do x <- getChar
          putChar x
          putChar x
```

Cuidado com a indentação: todas as ações devem estar alinhadas à mesma coluna.

Ações derivadas

Existem várias ações de I/O definidas no prelúdio a partir das mais simples.

Vamos ver algumas dessas funções e possíveis definições.

Funções *putStr* e *putStrLn*

A funções `putStr` do prelúdio imprime uma cadeia; pode ser definida usando `putChar` recursivamente.

```
putStr :: String -> IO ()
putStr []     = return ()
putStr (x:xs) = do putChar x
                   putStr xs
```

A função `putStrLn` imprime a cadeia acrescentado uma mudança de linha no final.

```
putStrLn :: String -> IO ()
putStrLn xs = do putStr xs
                 putChar '\n'
```

Função *getLine*

A função `getLine` lê carateres até a uma mudança de linha e retorna a cadeia de texto correspondente.

```
getLine :: IO String
getLine = do x <- getChar
             if x==’\n’ then
                return []
              else
                do xs <- getLine
                   return (x:xs)
```

Ações IO pré-definidas

```
putChar :: Char -> IO ()          -- escrever um carater

putStr :: String -> IO ()         -- escrever uma cadeia

putStrLn :: String -> IO ()       -- idem; muda de linha

print :: Show a => a -> IO ()       -- imprimir um valor

getChar :: IO Char                    -- ler um caracter

getLine :: IO String                    -- ler uma linha

getContents :: IO String    -- ler toda a entrada padrão
```

Combinando leitura e escrita

```
boasVindas :: IO ()
boasVindas = do
   putStr "Como te chamas? "
   nome <- getLine
   putStr ("Bem-vindo, " ++ nome ++ "!\n")
```

Jogo *Hi-Lo*

Exemplo maior: um jogo de adivinha

- o computador escolhe um número secreto entre 1 e 1000
- o jogador vai fazer tentativas de advinhar
- para cada tentativa o computador diz se é *alto* ou *baixo*
- a pontuação final é o número de tentativas

Jogo *Hi-Lo* (cont.)

```
Tentativa? 500
Demasiado alto!
Tentativa? 250
Demasiado baixo!
Tentativa? 350
Demasiado alto!
Tentativa? 300
Demasiado baixo!
Tentativa? 320
Demasiado alto!
Tentativa? 310
Acertou em 6 tentativas
```

Jogo *Hi-Lo* (cont.)

Vamos decompor em duas partes:

main escolhe o número secreto e inicia o jogo

jogo função recursiva que executa a sequência de perguntas e respostas do jogo

Programa

```
-- função para interações do jogo
jogo :: Int -> Int -> IO ()
jogo num cont  =
-- argumentos: número secreto e contador de tentativas
  do putStrLn "Tentativa? "
     str <- getLine
     let tent = read str  -- converte String -> Int
     if tent>num then
        do putStrLn "Demasiado alto!"
           jogo num (cont+1)
     else if tent<num then
             do putStrLn "Demasiado baixo!"
                jogo num (cont+1)
            else do putStr "Acertou em "
                    putStrLn (show cont ++ " tentativas")
```

Programa (cont.)

```
module Main where

-- biblioteca para números pseudo-aleatórios
import System.Random(randomRIO)

-- ponto de entrada do programa
main :: IO ()
main = do
   num <- randomRIO (1,1000) -- escolher um número
   jogo num 1      -- começar o jogo; contagem = 1
```

Programas completos

Se usarmos o GHC em vez do o GHCi podemos compilar o programa e obter um executável separado.

O ponto de entrada do programa completo é a ação `main` no módulo `Main`.

Para compilar e executar na *shell*:

```
$ ghc jogo.hs -o jogo
$ ./jogo
```