

Programação Funcional e Lógica

Introdução à Programação Funcional

Prof. Ricardo Couto A. da Rocha

`rcarocho@ufg.br`

UFG – Regional de Catalão

Roteiro

- Paradigma Imperativo vs. Funcional
- Características de Linguagens Funcionais
- Vantagens do Paradigma Funcional
- Exemplos Reais de Uso

Paradigmas de Linguagem

- **Imperativa**

- Características centrais são variáveis, declarações de atribuição, iteração
- Inclui as linguagens que suportam orientação a objetos
- Inclui linguagens de script
- **Exemplos:** C, Java, Perl, JavaScript, Visual BASIC .NET, C++

- **Funcional**

- Realiza computações aplicando funções aos parâmetros fornecidos
- **Exemplos:** LISP, Scheme, Haskell, ML

Paradigmas de Linguagem

- **Lógica**
 - Baseada em regras
 - Regras são especificadas sem uma ordem particular
 - **Exemplo:** Prolog
- Há ainda linguagens que são multiparadigma → oferecem construções de mais de um paradigma. **Exemplo:** Java (OO, procedural), Scala (OO, procedural, funcional)

Paradigma de Programação

- Especifica uma maneira particular de descrever uma computação ou a resolução de um problema computacional
- Uma computação ou solução é descrita usando construções linguísticas que não existem - ou não são diretamente mapeadas - em outro paradigma.

Metáfora: Sistema de Escrita

- Alfabeto romano ou latino
 - Base para diversas línguas como português, inglês, espanhol e francês.
- Conjunto de símbolos chamados letras
 - Combinados para formar palavras
 - Palavras possuem um significado próprio dependendo da língua e do contexto
 - Regras fonéticas determinadas a partir das combinações das letras (e as vezes da palavra)

Metáfora: Sistema de Escrita

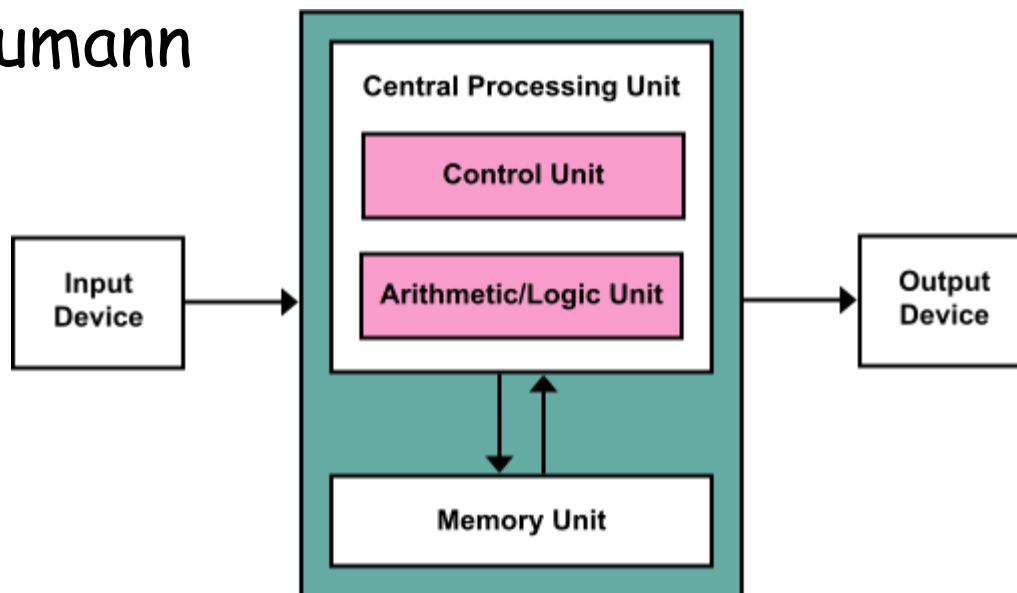
- Sílaba
 - Menor subdivisão fonética da palavra e formada por combinação de consoantes e vogais
 - Vogais são necessárias
- Dois tipos de letras que são combinadas
 - Consoantes: B, C, D, F, G, H, J, L, M, N P, Q, R, S, T, V, Z
 - Vogais: A, E, I, O, U
- Regras valem a grosso modo para todas as línguas baseadas nesse sistema de escrita
 - Sistema não define pronúncia ou significado

Sistema de Escrita Japonês

- Baseado no sistema chinês
 - Kanji: ideogramas que descrevem palavras (como verbos, substantivos, advérbios)
 - Hiragana: flexões de palavras e palavras funcionais como preposições.
 - Katagana: descrever palavras de origem não japonesa - descrição de natureza fonética
- Exemplo:
 - ラドクリフ、マラソン五輪代表に 1万m 出場にも含み
- Sistema não é baseado em um alfabeto
- Kanji: 50.000 símbolos → estudantes secundários devem saber 2.000.

Paradigma Imperativo

- Linguagens imperativas → seguem a arquitetura da máquina de Von Neumann



- Descrição de uma computação
 - Leitura ou escrita em endereço de memória
 - Execução de operações aritméticas sobre a memória (ou registradores)
 - Leitura de dispositivo de entrada ou escrita na saída

Exemplo: Quicksort

- Qual é a abordagem do Quicksort para ordenação de um conjunto de valores?
- Algoritmo
 - Escolher um elemento para ser o pivô
 - Dividir os elementos em duas partições
 - Colocar os elementos menores que o pivô em uma das partições
 - Colocar os elemento maiores que o pivô na outra partição
 - Repetir o algoritmo recursivamente para as duas partições
 - Unir as partições

Quicksort em C

```
void quicksort(int *A, int len) {  
    if (len < 2) return;  
  
    int pivot = A[len / 2];  
  
    int i, j;  
    for (i = 0, j = len - 1; ; i++, j--) {  
        while (A[i] < pivot) i++;  
        while (A[j] > pivot) j--;  
  
        if (i >= j) break;  
  
        int temp = A[i];  
        A[i]      = A[j];  
        A[j]      = temp;  
    }  
  
    quicksort(A, i);  
    quicksort(A + i, len - i);  
}
```

Quicksort em Haskell

```
qsort [] = []  
qsort (x:xs) = qsort [y | y <- xs, y < x]  
               ++ [x]  
               ++ qsort [y | y <- xs, y >= x]
```

Fibonacci

- Sequência dos primeiros N números de Fibonacci
 - $F_0 = 0$
 - $F_1 = 1$
 - $F_n = F_{n-1} + F_{n-2}$, if $n > 1$

Fibonacci em C

- Recursivo

```
long fibb(long a, long b, int n) {  
    return (--n>0) ? (fibb(b, a+b, n)):(a);  
}
```

- Iterativo

```
long int fibb(int n) {  
    int fnow = 0, fnext = 1, tempf;  
    while(--n>0){  
        tempf = fnow + fnext;  
        fnow = fnext;  
        fnext = tempf;  
    }  
    return fnext;  
}
```

Fibonacci em Haskell

- Versão recursiva tradicional

```
fib x =  
    if x < 1  
    then 0  
    else if x < 2  
         then 1  
         else fib (x - 1) + fib (x - 2)
```

- Versão sem recursão direta

```
fib n = go n 0 1  
    where  
        go n a b  
            | n == 0 = a  
            | otherwise = go (n - 1) b (a + b)
```

Roteiro

- Paradigma Imperativo vs. Funcional
- Características de Linguagens Funcionais
- Vantagens do Paradigma Funcional
- Exemplos Reais de Uso

Fundamentos

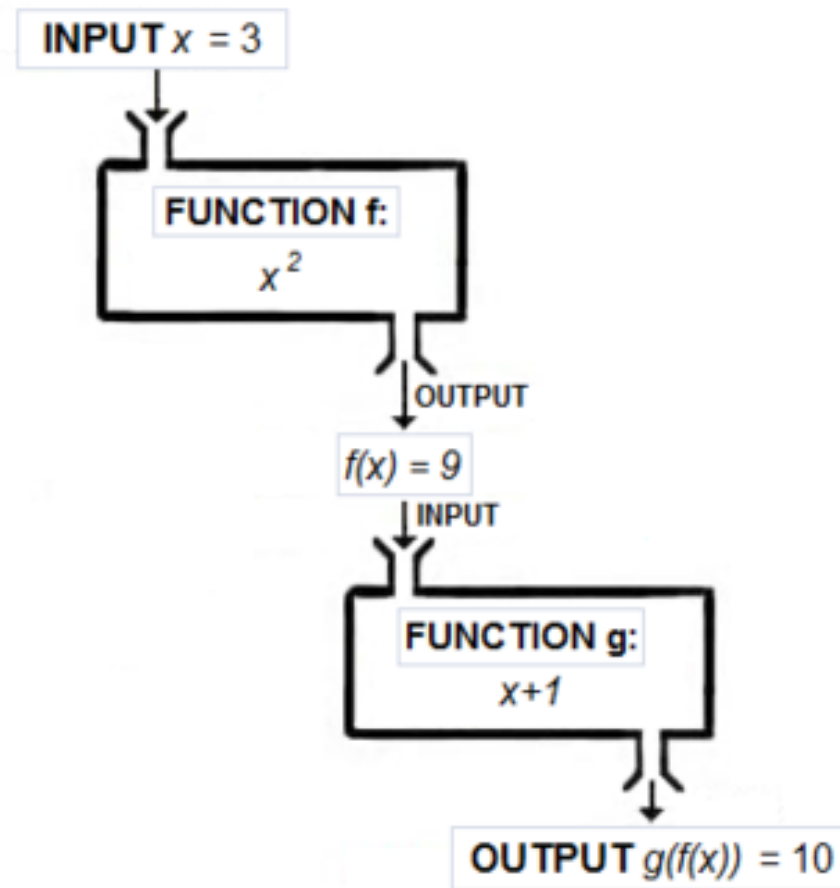
- **Objetivo** de projeto de linguagem funcional
 - Espelhar o mais próximo e amplamente possível as funções matemáticas
- Processo de computação em LF é **diferente** de linguagens imperativas
 - **Imperativa** → operações são realizadas e valores armazenados em variáveis (seu gerenciamento é fonte de complexidade)
 - **Funcional** → não há variáveis e não há estado da computação
- **Transparência referencial** → avaliação de função sempre produz o mesmo resultado para mesmos parâmetros

Características

- Não há estado
- Não há atribuição
- Não há variáveis
- Não há tipos (nem sempre)
- Funções são cidadãos de primeira classe → tudo é modelado/feito a partir de funções
- Linguagem puramente funcional
 - Não há relaxamento nas características acima

Computação em Programa Funcional

- Um programa em linguagem funcional, realiza computações por meio da avaliação de expressões.
 - Resultado do programa é o último valor avaliado



Linguagens Funcionais


- Uma linguagem funcional é chamada **pura**, quando não há nenhum relaxamento nos pressupostos do paradigma
 - **Tudo é imutável!** Variáveis ou estruturas de dados não mudam o seu valor → **transparência referencial**.
 - Expressões **não** tem **efeito colateral**.
 - Invocar uma função com os mesmos argumentos irá retornar sempre o mesmo valor.

Avaliação Preguiçosa (Lazy)

- Expressões não são avaliadas até que os seus resultados/valores são realmente necessários.
 - Em uma linguagem imperativa, todos os argumentos precisam ser avaliados antes que uma invocação ocorra.
- Permite a construção de abstrações elegantes e poderosas.
 - Exemplo: estruturas de dados infinitas

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
```

Comparação entre Linguagens Funcionais

Name ↕	Pure ↕	Lazy Evaluation ↕	Typing ↕	Abstract Data Types ↕	Algebraic Data Types ↕	Data is Immutable ↕	Type Classes ↕	Closures ↕
Lisp	No ^[1]	Yes ^[2]	Dynamic ^[3]	Yes ^[4]	?	No ^[5]	?	Yes ^[6]
Scheme	No ^[7]	No ^[7]	Dynamic ^[7]	Yes ^[8]	No native support ^[9]	No ^[10]	?	Yes ^[11]
Clojure	No ^[12]	Yes ^[13]	Dynamic ^[14]	Yes ^[15]	Yes ^[16]	Yes ^[17]	?	?
ML	No ^[18]	No ^{[19][20]}	Static ^[21]	?	?	Yes ^[22]	?	?
Lazy ML	Yes ^[23]	Yes ^[23]	Static ^[24]	?	?	Yes, by extension from ML ^[22]	?	?
OCaml	No ^[25]	Yes, by using the Lazy module ^[25]	Static ^[26]	Yes ^[27]	Yes ^[28]	Yes ^[29]	?	Yes ^[25]
F#	No ^[30]	Yes ^[31]	Static ^[32]	?	Yes "Discriminated Unions" 	Default ^[33]	?	Yes ^[34]
Haskell	Yes ^[35]	Yes ^[36]	Static ^[37]	Yes ^[35]	Yes ^[38]	Yes ^[39]	Yes ^[40]	Yes ^[41]
Frege	Yes ^[42]	Yes ^[42]	Static ^[42]	Yes ^[42]	Yes ^[42]	Yes ^[42]	Yes ^[42]	Yes ^[42]
Scala	No ^[43]	Yes, with the lazy keyword ^[44]	Static ^[43]	Yes ^[45]	Yes ^[45]	Default ^[46]	Yes ^[47]	Yes ^[45]
JavaScript	No ^[48] <small>[unreliable source?]</small>	Yes, with extension ^[49]	Dynamic ^[50]	Yes, with extension ^[51]	Yes, with extension ^[52]	Partial ^{[53][54]}	?	Yes ^[55]
Clean	Yes ^[56]	Yes, with optional strictness annotations ^[57]	Static with uniqueness/optionally dynamic ^[58]	Yes ^[57]	Yes ^[57]	Yes, except for unique types ^[57]	Yes ^[57]	Yes ^[57]
Miranda	Yes ^[59]	Yes ^[60]	Static ^[59]	Yes ^[61]	Yes ^[59]	?	?	?
SASL	Yes ^[62]	?	Dynamic ^[63]	?	?	?	?	?
Elixir	No	Yes, with the Stream module ^[64]	Dynamic	?	?	Yes	No	Yes
Erlang	No	No ^[65]	Dynamic	Yes ^[66]	Partial, only tuples and records ^[67]	Yes ^[68]	No	Yes
Elm	Yes	No	Static ^[69]	?	Yes ^[70]	Yes ^[69]	?	No
Python	No ^[71]	Partial (simulated using generators) ^[72]	Dynamic ^[73]	Yes ^[74]	?	Partial ^[75]	Yes ^[76]	Yes ^[77]
Candle	Yes ^[78]	?	Dynamic ^[79]	?	Partial, only Sequence and Map ^[80]	Partial ^[81]	?	?
Idris	Yes ^[82]	Yes ^[82]	Static ^[82]	Yes ^[82]	Yes ^[82]	Yes ^[82]	Yes ^[82]	Yes ^[82]
Kotlin	No	Yes (using Delegation)	Static	Yes	Partial, only Sequence and Map	Default	Yes	Yes

Roteiro

- Paradigma Imperativo vs. Funcional
- Características de Linguagens Funcionais
- Vantagens do Paradigma Funcional
- Exemplos Reais de Uso

Vantagens do Paradigma

- Expressividade
- Transparência referencial e ausência de efeito colateral
 - Programas são mais fáceis de testar e de garantir a corretude
 - Facilidade particularmente útil no desenvolvimento de programas **distribuídos, concorrentes ou paralelos.**

Roteiro

- Paradigma Imperativo vs. Funcional
- Características de Linguagens Funcionais
- Vantagens do Paradigma Funcional
- Exemplos Reais de Uso

Exemplos de Uso

- Facebook
 - **Erlang** no Facebook Chat
 - **Haskell** na filtragem de spams
 - Múltiplos projetos em **Ocaml**
- Amazon
 - **Erlang** em sistema de BD elástico
- Google
 - **Scala** em grandes projetos
- Ericsson
 - **Erlang** nos sistemas de telefonia celular (linguagem desenvolvida na empresa)

https://wiki.haskell.org/Haskell_in_industry