

Linguagens de Programação Funcionais

Prof. Dr. Eduardo Takeo Ueda
eduardo.tueda@sp.senac.br

Introdução (1/3)

- É uma categoria de linguagens não-imperativas
- Imperativas
 - Uso eficiente das arquiteturas de computadores de Von Neumann
- A arquitetura não deveria ser uma restrição no processo dedesenvolvimento de software
- Alguns outros paradigmas existem:
 - Não são eficientes
 - Não dominam o mercado

Introdução (2/3)

- O paradigma de programação funcional é baseado em funções matemáticas
- **LISP** iniciou como uma linguagem funcional pura, mas adquiriu algumas características imperativas
- Outras linguagens:
 - **Schema, Common LISP, ML, Miranda, Haskell**

Introdução (3/3)

- Uma linguagem funcional provê:
 - Conjunto de primitivas
 - Conjunto de formas funcionais para construir funções complexas a partir das funções primitivas
 - Operações de aplicação de funções
 - Alguma forma de armazenamento de dados

LISP

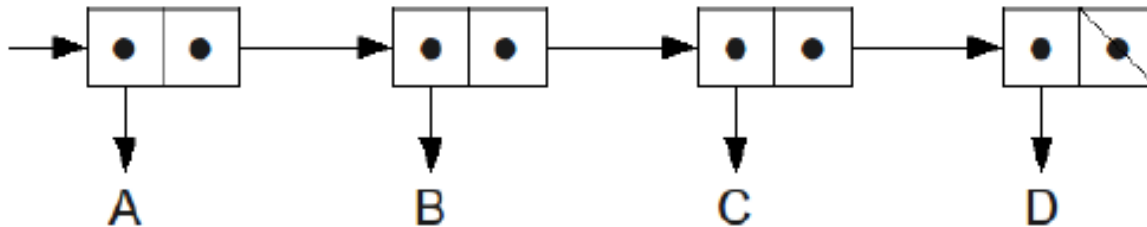
- Família de linguagens de programação concebida por John McCarthy em 1958
- A mais antiga e mais amplamente usada
- **LISP** – **List Processing** (a lista é a estrutura de dados fundamental desta linguagem)
- Com exceção da primeira versão, todos os dialetos incluem algumas características de linguagens imperativas:
 - Variáveis
 - Instruções de atribuição
 - Iteração

Tipos de Dados e Estruturas (1/3)

- **LISP** é uma linguagem sem tipos
- Dois tipos de objetos de dados:
 - Átomos:
 - São os símbolos de **LISP**
 - Listas:
 - São as estruturas de dados

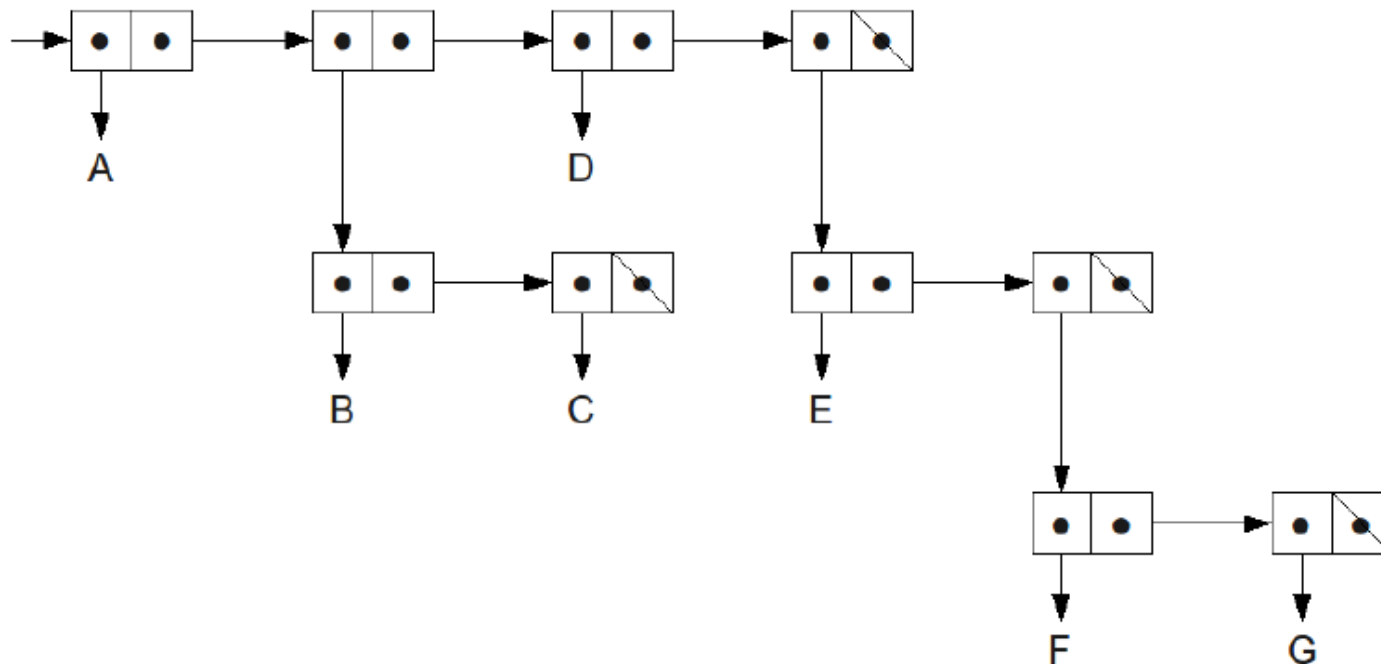
Tipos de Dados e Estruturas (2/3)

- Listas são especificadas delimitando seus elementos por parêntesis:
 - (A B C D)



Tipos de Dados e Estruturas (3/3)

- Estruturas de listas aninhadas são também permitidas:
- (A (B C) D (E (F G)))



Aplicações

- LISP é versátil e poderosa
- LISP foi desenvolvida para computação simbólica e aplicações de processamento de listas
- Editor de texto escrito em LISP: EMACS
- Cálculos simbólicos: MACSYMA
- Ensino introdutório de programação
- LISP na Inteligência Artificial:
 - Sistemas Especialistas
 - Representação do conhecimento
 - Aprendizado de máquina
 - Processamento de linguagem natural
 - Sistemas de treinamento inteligente
 - Modelagem da fala e visão

Comparação

- Linguagem Imperativa:
 - Gerência de variáveis e atribuição de valores
 - Eficiência
 - Construção de programas requer mais tempo e esforço

- Linguagem Funcional:
 - Não se preocupa com variáveis
 - Ineficiência
 - Programação em altíssimo nível:
 - Não requer muito tempo e esforço (principal vantagem)
 - Sintaxe muito simples:
 - Estrutura de listas
 - Semântica também simples:
 - Tudo são listas

Exemplo Introdutório de LISP

```
>14 ;calcule o valor deste símbolo
14
>'14 ;dê o símbolo sem calcular o valor
14
>(setq a 20) ;dê o valor 20 a variável 'a'
20
>a ;valor de a
20
>'a ;volte o símbolo sem calcular o valor
A
>(+ 1 2 3 4 5) ;dê a soma destes números
15
>(+ 1(- 12 6)) ;calcule
7
>(defun somasub(a b c)(+ a (- b c))) ;define uma nova função
SOMASUB ;responde que entendeu a nova função
>somasub(1 12 6)
7
>(exit) ;sai do LISP
```

Função Eval

- Se nada for dito, **LISP** supõe que todo átomo é o nome de uma função e para indicar que não é o caso, existe a função inversa do **EVAL**, chamado **QUOTE**, representada frequentemente por uma aspa simples:

```
>(a b c)
Error: the function A is not defined.
>'(a b c)
(A B C)
>(quote (a b c))
(A B C)
```

Seletores CAR e CDR

- **(car lista)**
 - Retorna o primeiro elemento de uma lista
- **(cdr lista)**
 - Retorna a lista sem o primeiro elemento

```
>(car '(a b c))  
A  
>(car '((a b) c d))  
(A B)  
>(car 'a)  
Error: bad argument type - A  
>(cdr '(a b c))  
(B C)  
>(cdr '())  
NIL  
>(car (cdr '(meu gato preto está dormindo)))  
GATO
```

Atribuição

- Corresponde ao que em outras linguagens é uma inicialização de variável, bom costume em muitas linguagens, tais como **Pascal**, **C++** e **Java**, e mau estilo em **LISP**

```
>(set 'a '(s d e))  
(S D E)  
>(car a)  
S  
>(car 'a)  
Error: bad argument type - A
```

Função Construtora

- Esta função tem por argumentos um átomo ou lista e coloca o átomo ou lista como primeiro elemento da nova lista

```
>(cons 'meu '(gato preto))  
(MEU GATO PRETO)  
>(cons '(a) '(b))  
((A) B)  
>(cons '(gato preto) '(sobre PC))  
((GATO PRETO) SOBRE PC)  
>(append '(a s) '(d e f))  
(A S D E F)
```

Predicados (1/3)

■ ATOM

- Retorna T se o que se segue é um átomo, e NIL caso contrário

```
>(set 'a '(s d e))
```

```
(S D E)
```

```
>(atom a)
```

```
NIL
```

```
>(atom 'a)
```

```
T
```

```
>(atom ())
```

```
T
```

```
>(atom '())
```

```
T
```

```
>(atom '(()))
```

```
NIL
```


Predicados (2/3)

■ NULL

- Retorna T se o que se segue é NIL, e NIL caso contrário

```
>(set 'a '(s d e))
```

```
(S D E)
```

```
>(null a)
```

```
NIL
```

```
>(null 'a)
```

```
NIL
```

```
>(null '())
```

```
T
```

```
>(null (cdr '(b)))
```

```
T
```

Predicados (3/3)

■ EQL

- Determina se dois elementos atômicos são iguais
- Dado listas por argumento retornará NIL, mesmo que as listas sejam iguais

```
>(set 'a '(s d e))  
(S D E)  
>(eq1 (car a) 's)  
T  
>(eq1 '(a s d) '(a s d))  
NIL  
>(eq1 'ram 'rom)  
NIL  
>(eq1 a '(s d e))  
NIL
```

Condicional

```
(cond (causa_1 consequencia_1)
      (causa_2 consequencia_2)
      (causa_n consequencia_n))
```

```
>(setq y 70)
```

70

```
>(cond ((> y 65) (setq j 4))
      ((> y 21) (setq j 3))
      ((> y 10) (setq j 2))
      (T       (setq j 1)))
```

4

Funções (1/5)

```
(defun nome (arg_1 arg_2 arg_n)
  tarefa_1
  tarefa_2
  tarefa_n)
```

```
>(defun fatorial(n)
  (cond ((= n 0) 1)
        (T (* n (fatorial(- n 1))))))
```

FATORIAL

```
>(fatorial 3)
```

6

```
>(fatorial 10)
```

3628800

Funções (2/5)

```
>(defun pertence(elt lst)
  (cond ((null lst)          nil)
        ((eq1 elt (car lst)) T)
        (T                  (pertence elt (cdr lst)))))

PERTENCE
>(setq lista '(a s d f g))
(A S D F G)
>(pertence 'd lista)
T
>(pertence 'x lista)
NIL
```

Funções (3/5)

```
>(defun quadrado (x)  
    (* x x))
```

QUADRADO

```
>(quadrado 5)
```

25

Funções (4/5)

```
>(defun conta(lst)
      (cond ((null lst) 0)
            (t (+ 1 (conta (cdr lst))))))
```

CONTA

```
>(conta '(a b c))
```

3

Funções (5/5)

```
>(defun soma(lst)
  (cond ((null lst) 0)
        (t (+ (car lst) (conta (cdr lst))))))
```

SOMA

```
>(soma '(1 2 3))
```

6