# Introdução à Programação Funcional

PPGIA - PUCPR

Prof. Fabrício Enembreck

1

# Conteúdo

- Introdução ao Cálculo Lambda e à Programação Funcional
- Introdução ao LISP e ao Common Lisp
- Funções Lambda e binding
- Funções de seqüências
- Listas e funções recursivas
- Grafos em LISP

2

# Common Lisp

- de origem acadêmica, LISP desenvolveu-se de maneira desorganizada
- Vários dialetos: Interlisp, Franzlisp, Maclisp, Zetalisp, Scheme, Le_Lisp
- 1984, publicação de *Common Lisp: The Language,* padronização que reuniu em um único dialeto as características mais interessantes dos dialetos mais utilizados (principalmente Maclisp, Zetalisp, Scheme e Interlisp)
- Todas as implementações apresentam uma shell.

3

# Elementos Fundamentais da Linguagem

- Tudo em LISP é S-expressão (único tipo).
- Variáveis não têm tipos, mas constantes têm tipo.
- Qualquer variável aceita qualquer S-expressão como valor!
- Contudo, há alguns subtipos.
  - S-expressões:
    - átomos:
      - numéricos
      - Strings
      - Símbolos
    - cons:
      - Listas
      - não-listas
- Representação de S-expressões:
  - representação gráfica: caixinhas
  - representação textual: caracteres

4

# Representação Gráfica e Textual

- Átomos têm representação gráfica IGUAL à representação textual

| | Gráfica | Textual |
|---|---|---|
| número | 1 | 1 |
| string | "casa" | "casa" |
| símbolo | CASA | CASA |

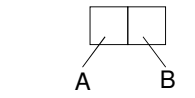
- Dentro da string faz diferença Maiúsculas / Minúsculas. Em símbolos tanto faz. O interpretador sempre imprime símbolos usando maiúsculas.

5

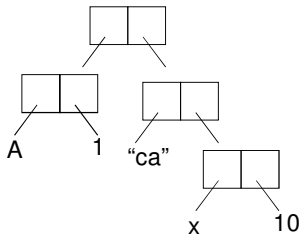
# Representação Gráfica e Textual – Cons

**Cons**: estrutura formada por um par de S-expressões. Ou seja, duas S-expressões são usadas para formar uma. Ex: podem ser dois átomos; uma pode ser átomo e a outra par-com-ponto; ambas podem ser pares-com-pontos; etc

Gráfica (Caixas) Textual




**(A . B)**




**((A . 1) . ( “ca” . (x . 10)))**

6

# Representação Gráfica e Textual – Listas

- Átomo especial: NIL (do latim nada): Representa a lista vazia

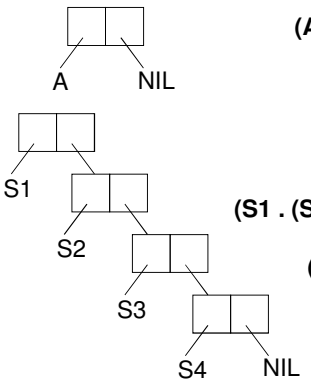
| Gráfica | Textual |
|---|---|
| NIL | NIL ou () : formas equivalentes |

7

# Representação Gráfica e Textual – Listas (cont.)

**Lista (definição)**: Uma lista é NIL ou um cons cujo segundo elemento é uma lista.

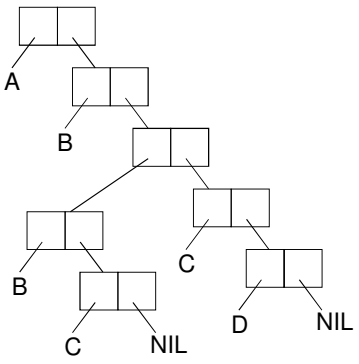
Gráfica (Caixas)




Textual

**(A . NIL) ou (A)**

**(S1 . (S2 . (S3 . (S4 . NIL))))**
**ou**
**(S1 S2 S3 S4)**




**(A . (B . ((B . (C . NIL)) . (C . (D . NIL)))))**
**ou**
**(A B (B C) C D)**

8

## Cons-truindo Listas

- (cons elem lista) retorna uma cópia de lista com elem inserido como seu primeiro elemento
- Exemplo:

```
> (cons 'a '(b c))
(A B C)
> (cons 'a '(b))
(A B)
> (cons 'a nil)
(A)
```

- Teorema: (cons (car L) (cdr L)) = L

9

## Cons

- O que acontece se passarmos um átomo como segundo argumento de cons?

```
> (cons 'a 'b)
(A . B)
```

- De onde veio o ponto entre A e B?
- Resposta:
  - O ponto sempre existe num cons, mas nem sempre é impresso
  - Para entender, precisamos rever um conceito anterior ...

10

# Conses (*S-Expressions*)

- Em LISP, se algo não é um átomo, então é um **cons** ou "*S-expression*"
- Um **cons** nada mais é que um registro com dois campos, o primeiro é chamado de **car** e o segundo de **cdr**
- A regra do ponto:
  - O **cons** é escrito com o valor dos dois campos entre parênteses ou separados por um ponto
  - *Entretanto*, se o campo **cdr** é **nil** ou um **cons**, o ponto pode ser omitido

11

# Conses – Mais Exemplos

```
> (cons 'a 'b)
(A . B)
> '(a . b)
(A . B)
> '(a . nil)
(A)
> '(a . (b . (c . nil)))
(A B C)
> '(a . (b . c))
(A B . C)
> '((a . b) . c)
((A . B) . C)
> '((a . b) . (b . c))
((A . B) B . C)
```

12

# Conses e Listas

- Podemos ver então que listas são conses que nunca são escritos com pontos
  - Muitos autores preferem ignorar conses que não são listas
- Afinal, o que é então uma lista?
- Eis uma resposta formal:
  - Axioma: nil é uma lista
  - Teorema: Se L é uma lista, e elem é um átomo ou uma lista, então (cons elem L) é uma lista

13

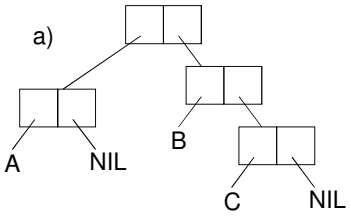
# Exercícios

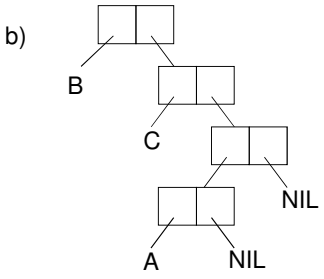
- 1. Desenhe a representação gráfica das seguintes S-expressões:
  - ( (A . B) . (C . NIL) )
  - (A . (B . (C . D)))
  - ((((A . B) . C) . D) . NIL)

14

# Exercícios

- 2. Escreva a representação textual (lista e cons) das seguintes S-expressões:




15

# Exercícios

- 3. Descreva o que acontece quando as expressões abaixo são avaliadas:
  - a. (list 1 (+ 2 3))
  - b. (if (listp 1) (+ 1 2) (+ 3 4))
  - c. (list (and (listp 3) t) (+ 1 2))
  - d. (+ (- 5 1) (+ 3 7))
- 4. De três expressões "cons" distintas que retornam (a b c).
- 5. Usando "car" e "cdr", defina uma s-expressão que retorna o quarto elemento de uma lista.
- 6. Defina uma s-expressão que retorne o maior entre dois números.

16

# Introdução à Programação Funcional e ao Common Lisp

- Sintaxe

  A expressão matemática *z = f(x,y)* é escrita em LISP da seguinte maneira:

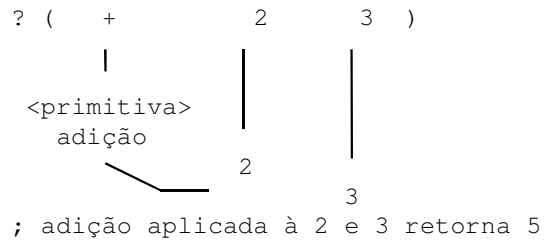
  (f x y)

  A avaliação da função retorna o valor *z*.
- Sintaxe para Listas : “(”, “)” e espaço.
  - Ex.: (+ 2 (+ 3 5)) é uma expressão bem formada.
- Primitivas: Common Lisp comporta +-1000 funções primitivas

17

# Introdução à Programação Funcional e ao Common Lisp (cont.)

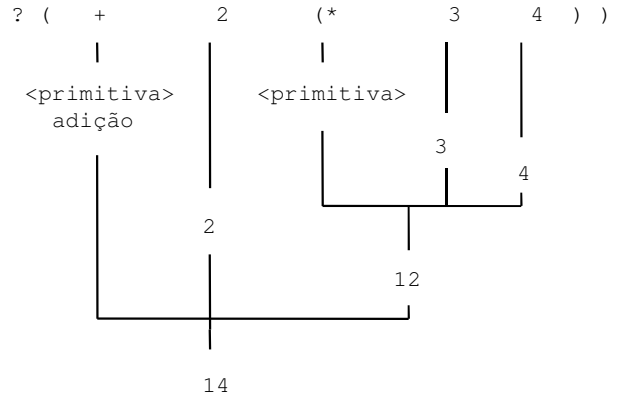
- Exemplo de Avaliação de uma expressão




18

# Introdução à Programação Funcional e ao Common Lisp (cont.)

- Exemplo de Avaliação Recursiva de uma expressão: (+ 2 (* 3 4))




19

# Introdução à Programação Funcional e ao Common Lisp (cont.)

- Para evitar a avaliação de uma expressão utiliza-se “ ’... ” ou “ (quote ...) ” .
- Ex.

?> ‘(funcao 2 (funcao1 (funcao2 x y) valor) a)
(FUNCAO 2 (FUNCAO 1 (FUNCAO2 X Y) VALOR) A)

?> (quote (funcao 2 (funcao1 (funcao2 x y) valor) a))
(FUNCAO 2 (FUNCAO 1 (FUNCAO2 X Y) VALOR) A)

20

# Criação de variáveis especiais (globais)

- **set**

  Nome de um Símbolo

  ?> (set ‘var “Joao”)
- **setq**

  ?> (setq var “Joao”)
- **setf**

  ?> (setf var “Joao”)

?> var
“Joao”

?> (**symbol-value** ‘var)
“Joao”

**Uma Variável criada com atribuição é sempre um ponteiro para um símbolo, string, número ou ref. a um objeto!!!**

21

# Variáveis globais

- Notação default: *nome-da-variavel*

?> (defparameter *var* ‘(uma lista de elementos))
?> (defvar *var* ‘(uma lista de elementos))
?> (setq *var* ‘(uma lista de elementos))

22

# Manipulação de símbolos

- Um símbolo possui três elementos:
  - Valor de impressão
    - usado para mostrar o valor de um símbolo
  - Um pacote de origem
    - Define o escopo da existência do símbolo
  - Uma lista de propriedades
    - plist do símbolo usado para armazenar/recuperar valores. Ex: (setf (symbol-plist 'm) '(:cor azul :local nil))

23

# Manipulação de símbolos

```
?> (setq joao "João da Silva")
"João da Silva“

?> (setf (symbol-plist 'joao) '(:nome "João" :sobrenome "Silva" :cargo
    programador))
(:NOME "João" :SOBRENOME "Silva" :CARGO PROGRAMADOR)

?> (get 'joao :cargo)
PROGRAMADOR

?> (remprop 'joao :sobrenome)
T

?> (symbol-plist 'joao)
(:NOME "João" :CARGO PROGRAMADOR)
```

Propriedades são definidas em pares!!!

```
?> (symbol-package 'joao)
#<The COMMON-GRAPHICS-USER package>

?> (symbol-value ‘joao)
"João da Silva“
```

*Em algumas distribuições de LISP a função plist é chamada *symbol-plist*

24

# Mais sobre variáveis e símbolos

?> (set ‘var “Joao”)  é equivalente a
?> (set (quote var) “Joao”)

- Um símbolo pode estar ligado ou não. Ex.:

?>  (boundp ‘var)
T

- Um símbolo não é uma string

?> (setq var ‘Joao)
JOAO

é diferente de

?> (setq var “Joao”)
“Joao”

25

# Manipulação de Listas

- Listas

?> (setq var ‘(o maior ataque))
(O MAIOR ATAQUE)

Esses elementos são símbolos, não strings!!!

- Manipulação de listas:

car, cdr, last, cons, list, append

26

# Manipulação de Listas

?> (**setq** var ‘(o maior ataque))
(O MAIOR ATAQUE)

?> (**car** var)  ;; o primeiro elemento de var
O

?> (**cdr** var)  ;; uma lista com o corpo de var
(MAIOR ATAQUE)

?> (**last** var)  ;; uma lista com o último elemento de var
(ATAQUE)

?> (**cons** ‘tenho var)  ;; uma lista com a cabeça ‘*tenho* e corpo *var*
(TENHO O MAIOR ATAQUE)

27

# Manipulação de Listas

?> (**list** ‘a ‘b ‘c ‘(e f))  ;; constrói uma lista com os elementos fornecidos
(A B C D (E F))

?> (**append** ‘(a b c) ‘(d e f))  ;; concatena duas listas
(A B C D E F)

- Abreviações
  (car (car (car x))) é igual a (caaar x)
  (cdr (cdr (cdr x))) é igual a (cddr x)
  (car (cdr (cdr (car (car x))))) é igual a (caddaar x)

28

# Manipulação de Listas - Predicados

- Valores lógicos

```
?> t
T

?> nil
NIL

?> (or nil nil nil nil)
NIL

?> (or nil t nil nil)
T

?> (and t t t)
T
```

```
?> (numberp 34)
T

?> (listp '(or t t))
T

?> (floatp 4.567)
T

?> (integerp 2.31)
NIL

?> (null nil)
T

?> (symbolp "Albert")
NIL

?> (stringp 'Albert)
NIL
```

29

# Criação de variáveis dinâmicas (binding)

- (let (<declarações>) corpo)

```
(let ((v 5) (v1 10) v2)
   (print v2)
   (setq v2 10)
   (list (+ v v1) v2))

NIL
(15 10)
```

Declarações

Corpo

O valor de um binding é sempre uma cópia, não um ponteiro!!!

30

# Binding (Variáveis Locais)

- (let* (<declarações>) corpo)

```
(let* ((v 5) (v1 v) v2)
   (print v2)
   (setq v2 10)
   (list (+ v v1) v2))
```

Com **let*** as variáveis são inicializadas em série!!!

```
NIL
(10 10)
```

31

# Iterações

```
?> (setq lista '(cada vez maior))
?> (dolist (l lista retorno) (print l))
CADA
VEZ
MAIOR
```

opcional

```
?> (dotimes (cont 5 retorno) (print cont))
0
1
2
3
4
```

32

# Iterações

(loop

...

(return) ou (return retorno)

...

)

33

# Definição de Funções

(defun <nome> <lista de parâmetros>

<corpo da função>)

#| ... |# : Comentário de bloco

; ... : Comentário de linha

“ ...” : Comentário de bloco dentro da função

34

# Funções (cont.)

- Exemplo

(defun cubo (y)

"A função cubo calcula o cubo de um número. Exemplo: (cubo 9) retorna 729"

(* y y y))

O valor retornado em uma função é sempre o valor da ***última*** expressão do corpo da função

35

# Exercícios

- Escreva as seguintes funções:
  - A) Que retorne o maior entre 3 números
  - B) Que encontre o maior elemento dentre uma lista de números
  - C) Que conta a quantidade de elementos numéricos de uma lista
  - D) Que cria uma lista de saída a partir dos elementos que estão nas posições ímpares de uma lista de entrada. (Listas iniciam na posição 0)
  - E) Que dados 2 valores inteiros representando um intervalo, imprima os inteiros que fazem parte desse intervalo em ordem crescente

36

# Modificadores de Parâmetros

- Diretiva &optional
  - Todos os parâmetros à direita de &optional são opcionais.
  - EX: ?> (defun funcao1 (n &optional n1)
    (print n)
    (if n1 (print n1)))

```
?> (funcao1 10)
10
nil
```

```
?> (funcao1 10 20)
10
20
20
```

37

# Modificadores de Parâmetros

- Diretiva &rest
  - Todos os parâmetros à direita de &rest são colocados dentro de uma lista.
  - EX: ?> (defun funcao1 (n &rest l1)
    (print n)
    (print l1))

```
?> (funcao1 10 20 30)
10
(20 30)
(20 30)
```

38

# Modificadores de Parâmetros

- Diretiva &key
  - Todos os parâmetros à direita de &key devem ser referenciados por um nome específico na chamada da função.
  - EX: ?> (defun funcao1 (n &key segundo terceiro)
    (print n)
    (print segundo)
    (print terceiro))

?> (funcao1 10 :terceiro 30 :segundo 20)
10
20
30
30

39

# Modificadores de Parâmetros

- As diretivas *optional*, *rest* e *key* podem ser usadas em conjunto
- Parâmetros também podem ter valores defaults
- Ex.:

```
?> (defun funcao1 (n &optional &key (segundo 22) (terceiro 37))
        (print n)
        (print segundo)
        (print terceiro))

?> (funcao1 1 :terceiro 20)
1
22
20
20
```

40

# Funções Anônimas

(lambda <lista de parametros>
<corpo da função>)

?> ((lambda (x y) (list x y)) 10 20)
(10 20)

***lambda*** define uma expressão λ, que é um tipo especial de função: uma função sem nome

41

# Construções Dinâmicas de Função

?> (setq fn (list ‘lambda ‘(x) ‘(list x x)))
(lambda (x) (list x x))

?> (**eval** (list fn 10))
(10 10)

Uma função é manipulada como um tipo de dado qualquer!!!

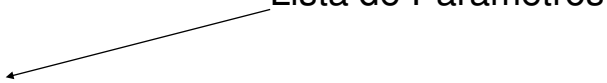
42

# Aplicação de Funções

```
?> (defun ff (x y) (* x y))
FF

?> (apply ff '(2 3))
6

?> (funcall ff 2 3)
```




43

# Blocos Condicionais

**If**

```
(if p
     Bloco 1
Bloco 2)
```

**when**

```
(when p a b c)  <=> (and p (progn a b c))
                    (cond (p a b c d))
                    (if p (progn a b c))
                    (unless (not p) a b c)
```

**unless**

```
(unless p a b c) <=>(when (not p) a b c)
```

44

# Blocos Condicionais (cont.)

**cond**

```
(cond (p1 e1 e'1 e"1 ...)
      (p2 e2 e'2 e"2 ...)
      ...
      (pn en e'n e"n ...))
```

**case**

```
(case keyform
    (key-list-1 e1 e'1 e"1 ...)
    (key-list-2 e2 e'2 e"2 ...)
    ...
    (key-list-n en e'n e"n ...))
```

*Exemplo :*

```
(case input-code
  (a "a")
  (b "b")
  (c "c")
  (otherwise "unknown code..."))
```

45

# Exercícios

- 1) Implemente versões **recursivas (quando possível)** e **iterativas** das seguintes funções LISP:
  - ?> (maior 3 1) ;; retorna o maior entre 2 números
  - ?> (potencia Base Expo) ;; calcula a potência
  - ?> (n_elementos ‘(a b c)) ;; calcula o tamanho da lista
    3
  - ?> (n_esimo 2 ‘(a b c)) ;; encontra o n-ésimo elem.
    B
  - ?> (merge ‘(1 4 12 28) ‘(6 7 10)) ;; faz o merge
    (1 4 6 7 10 12 28)

46

# Exercícios

- ?- (insere ‘a 2 ‘(1 2 3)). ;; insere um elemento na posição
  (1 A 2 3)
- ?> (concatena ‘(a b c) ‘(1 2 3)) ;; une duas listas
  (A B C 1 2 3)
- ?> (inverte ‘(a b c)) ;; inverte uma lista
  (C B A)
- ?> (ultimo ‘(a b c)) ;; encontra o último elem. da lista
  C

47

# Exercícios

- 2) O que estas funções fazem?

```
(defun enigma (x)
    (and (not (null x))
        (or (null (car x))
            (enigma (cdr x)))))

(defun mystery (x y)
    (if (null y)
        nil
        (if (eql (car y) x)
            0
            (let ((z (mystery x (cdr y))))
              (and z (+ z 1))))))
```

48

# Exercícios

- 3) O que poderia ser colocado no lugar de "x" nas expressões abaixo para que o valor proposto seja encontrado?

a) ?> (car (x (cdr '(a (b c) d))))
B

b) ?> (x 13 (/ 1 0))
13

c) ?> (x #'list 1 nil)
(1)

49

# Exercícios

- 4) Usando os operadores apresentados em aula, defina uma função que receba uma lista como um argumento e retorne "true" se um dos elementos é uma lista.
- 5) Dê uma definição iterativa e uma recursiva de uma função que:
  - a) recebe um inteiro positivo e imprima o mesmo número de pontos.
  - b) recebe uma lista e retorna o número de vezes que o símbolo "a“ aparece nela.

50

# Exercícios

- 6) Usando os operadores apresentados em aula, defina uma função que receba uma lista como um argumento e retorne "true" se um dos elementos é uma lista.
- 7) Crie versões recursivas e iterativas de uma função que soma todos os elementos não nulos de uma lista de números.
- 8) Modifique a função a seguir de forma que ela some todos os elementos não nulos de uma lista de números corretamente

```
(defun somalista (lst)
    (let ((x (car lst)))
      (if (null x)
          (somalista (cdr lst))
          (+ x (somalista (cdr lst))))))
```

51