Introdução à Programação Funcional

PPGIA - PUCPR
Prof. Fabrício Enembreck

1

Conteúdo

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

2

Função Lambda

- Uma função lambda é uma função sem nome
- Normalmente uma ***função lambda*** está associada com uma função de ***mapeamento***

```
(defun funcao (lista)
   (if (> (length lista) 0)
      (mapcar #'(lambda (elem) (if (listp elem) elem)) lista)
    (progn
      (print "Lista vazia")
      nil)
   )
 )
```

função de mapeamento

função lambda

3

Funções de Mapeamento

- Objetivo
  - Simplificar o processamento de cada elemento de uma lista
- Estrutura de chamada

(funcao-mapeamento <função a executar> <lista>)

Função Lambda ou nome de uma outra função

Lista de elementos a ser processada

4

Funções de Mapeamento (cont.)

- (mapcar <f> <L>)
  - *Retorna uma lista com o mesmo tamanho de L, onde cada elemento corresponde ao resultado da aplicação da função "f" sobre o respectivo elemento de L*
  - Ex.: ?> (mapcar #'length '((a b) (a b c) (a)))
    (2 3 1)

5

Funções de Mapeamento (cont.)

- (mapc <f> <L>)
  - *Como mapcar mas o valor de retorno é L. Utilizada normalmente para formatar saída no vídeo.*
  - Ex.: ?> (mapc #'(lambda (x) (print (length x))) '((a b) (a b c) (a)))
    2
    3
    1
    ((A B) (A B C) (A))

6

Funções de Mapeamento (cont.)

- (maplist <f> <L>)
  - *Aplica a função "f" n vezes (n é o tamalho de "L") primeiro sobre a lista inteira, depois sobre o nth cdr de "L". Retorna o valor da aplicação da função sobre os elementos de "L".*
  - Ex.: ?> (maplist #'(lambda (x) (print x) (print (length x))) '(a b c))

```
(A B C)
3
(B C)
2
(C)
1
(3 2 1)
```

7

Funções de Mapeamento (cont.)

- (mapl <f> <L>)
  - *Mesmo que "mapl", mas o retorno é L.*
  - Ex.: ?> (mapl #'(lambda (x) (print x) (print (length x))) '(a b c))

```
(A B C)
3
(B C)
2
(C)
1
(A B C)
```

8

Macros

- Uma macro serve para construção dinâmica de funções
- Diferenças entre macros e funções:
  - Em macros os parâmetros não são avaliados na chamada
  - Uma macro é avaliada duas vezes: a primeira para reconstruir o corpo da macro e a segunda para avaliar o corpo construído da macro
  - Uma macro possui apenas um comando no nível zero

9

Macros (cont.)

- Exemplo de criação de uma macro

```
(defmacro meu-while (teste corpo)
        (list 'loop (list 'unless teste '(return nil)) corpo))
MEU-WHILE
?> (setq n 0)
?> (meu-while (< n 3) (progn (print n) (incf n)))
0
1
2
NIL
```

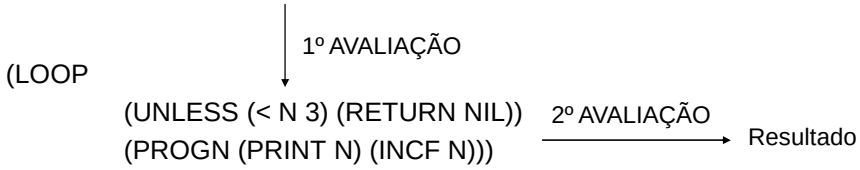
10

Macros (cont.)

- Exemplo de avaliação da macro meu-while

```
(defmacro meu-while (teste corpo)
          (list 'loop (list 'unless teste '(return nil))
    corpo))
```

?> (meu-while (< n 3) (progn (print n) (incf n)))



11

Macros (cont.)

- Verificando a expansão de uma macro (1ª avaliação)

?> (macroexpand '(meu-while (< n 3) (progn (print n) (incf n))))

T
(BLOCK NIL
 (TAGBODY
  #:G512 **(PROGN (UNLESS (< N 3) (RETURN NIL)) (PROGN (PRINT N) (INCF N)))**
      (GO #:G512)))
T

12

Macros (cont.)

- Notação utilizada para simplificar o corpo de macros:
  - “ ` ”, “ , ” e “ @ ”

```
(defmacro meu-while (teste &rest corpo)
    `(loop (unless ,teste (return nil))
          ,@corpo
          )
)
```

Indica que todos os parâmetros, a partir do segundo serão colocados na lista “corpo”

` : Introduz um texto que representa uma expressão LISP
, : Recupera o valor de uma avaliação (só pode ser usado dentro de um `)
@: Recupera todos os elementos de uma lista sem o parêntese inicial e final (só pode ser usado sobre uma lista e dentro de um `)

13

Macros (cont.)

```
(defmacro meu-while (teste &rest corpo)
    `(loop (unless ,teste (return nil))
          ,@corpo
          )
  )
```

Agora não precisamos do “progn” Devido à utilização do “&rest”

```
?> (meu-while (< n 3) (print n) (incf n))
```

14

Macros (cont.)

- Exemplos:
  - ?> (setq l ‘(a b c))
  - ?> `,l
    (A B C)
  - ?> `’,l
    ‘(A B C)
  - ?> `(list l)
    (LIST L)
  - ?> `(list ,l)
    (LIST (A B C))
  - ?> `’,@l
    (QUOTE A B C)
  - ?> `(c ,@l)
    (C A B C)
  - ?> `(c ,l)
    (C (A B C))

15

Exercícios

Implemente as seguintes funções LISP usando funções de mapeamento:

- ?> (soma-numero 10 ‘(0 5 10)) ;; soma um número aos elementos da lista
  (10 15 20)
- ?> (n_numeros ‘(a 10 3 f)) ;; retorna a quantidade de números existente em uma lista
  2
- ?> (formata ‘(a b c)) ;; formata* os elementos da lista
  - (“E-A” “E-B” “E-C”)

* Pesquisar a sintaxe da função “format”

16

Exercícios

- Implemente as seguintes macros LISP:
- ?> (concat (a b c) (1 2 3)) ;; une duas listas
  (A B C 1 2 3)

- ?> (meu-for (i 1) (< i 3) (print i) (incf i))

  1
  2 ← Valores impressos pela macro

  3 ← Valor retornado pela macro

17

Seqüências

- Listas e Arrays são derivações de Sequências
- LISP possui um grande conjunto de funções para manipular seqüências.

Função	Significado
length	Retorna o tamanho da seq.
subseq	Retorna uma parte da sequência
replace	Substitui um elemento
count	Conta o n. de ocorrências de 1 elemento
reverse	Inverte a ordem da seqüência
nreverse	Inverte e altera a ordem da seqüência
concatenate	Concatena duas ou mais seqüências
position	Encontra a posição de um elem.
find	Encontra um elemento na seq.
sort	Ordena a seqüência
search	Encontra uma subseq. em uma seq.
remove	Rem. o elem. e retorna uma cópia da seq. Original
delete	Remove o elemento e modifica a seq. original
delete-duplicates	Remove duplicações
substitute	Substitui um elemento
nth	Retorna o elemento da lista da pos. desejada
aref	Retorna o elemento da seqüência da pos. desejada
nsubstitute	Substitui um elemento e altera a seq. original
read-from-string	Transforma uma string numa expressão simb. LISP

18

Seqüências

- Modificadores comuns em funções de seqüências
  - (remove item sequence **:test** #'equal)
  - (find item sequence **:test** #'eq **:key** #'car)
  - (position #\b "foobar" **:start** 2 **:end** 5) → 3
  - (position #\b (subseq "foobar" 2 5)) → 1
  - (position 'a '(a b c) **:from-end** t) → 0

19

Exercícios

- 1) Dada a seguinte definição:

```
(setf mylist '((fhc frango)(collor tereza)
      (voce nao eh a mamae)
      (e por favor) (spielberg (toni ramos))
      (polvo lula mariscos)))
```

- Construa as listas abaixo usando qualquer função que você tenha aprendido até agora.
  - A) (frango nao por favor)
  - B) ((tereza collor) nao eh a mamae)
  - C) (collor fhc lula)
  - D) (spielberg nao eh a favor)

20

Exercícios

- 2) Escreva uma função “achata", que recebe uma única lista como parâmetro e retorna uma lista simples contendo todos os seus átomos:

Ex. (achata '(1 (2 (3))(4 5))) ==> (1 2 3 4 5)
(achata '(((((teste)))))) ==> (teste)

- 3) Escreva uma função REVERSO, que inverte os elementos de uma lista.

Ex. (reverso '(A (B C) D (E F))) ==> ((E F) D (B C) A)

- 4) Escreva uma função REVERSO-TOTAL, que inverte os elementos de uma lista e inverta também, recursivamente, os elementos das listas contidas dentro da primeira.
- Ex. (reverso-total '(A (B C) D (E F))) ==> ((F E) D (C B) A)

21

Exercícios

- 5) Defina funções LISP que resolvam os seguintes problemas. Para muitos deles já existem funções pré-definidas que fazem o serviço. Neste caso, o nome da função pré-definida é dado entre parêntesis.
  - A) Calcule o quadrado de n apenas fazendo somas.
  - B) Retorne uma lista com o último elemento de uma lista dada (last).
  - C) Receba um número variável de parâmetros (um ou mais) e construa uma lista com todos eles (list).
  - D) Retorna a concatenação de duas listas dadas como parâmetros (append).

22

Exercícios

- EX. 5 (continuação)
  - F) Verifique se uma lista é sublista da outra, isto é, consiste de um trecho consecutivo da outra.
  - G) Retorne a sublista contendo os n primeiros elementos de uma lista dada.
  - H) Retorne o último token de uma string que contém uma frase. Ex: “meu carro quebrou” → “quebrou”
  - I) Substitua parte de uma string por uma outra string. Ex: (substituir “xx” “as” “assim”) → “xxim”
  - J) Faça a intersecção de duas listas (intersection)
  - Retorne a parte de uma string que está entre duas outras partes.

  Ex: (remove-parte “casabrancadosul” “casa” “do”) → “branca”
  - Substitua a parte de uma string que está entre duas outras partes.

  Ex: (remove-parte “casabrancadosul” “casa” “do” “azul”) → “casaazuldosul”

23