Universidade Federal de São Carlos

Programa de Pós Graduação em Ciência da Computação - PPGCC

Paradigmas de Linguagem de Programação

Relatório do Trabalho
Desenvolvido na Linguagem Lisp

Alunos:

Bruno César Sales Alves

Patrícia Deud Guimarães

Docente Responsável: Heloísa Arruda Camargo

São Carlos - SP

17/06/2019

1. Exercício 01:

```
(defun Desparentize (L)
    (cond ((null L) nil)
    ((atom L)(list L))
    ((and (listp L) (equal nil (car L)))
                (cons nil (Desparentize(cdr L))))
        (t (append (Desparentize(car L)) (Desparentize (cdr L))))))
;(print (desparentize '( (1 (2) 3) 3)))

(defun conta-elem (elem lista)
    (
     cond ((null lista) 0)
          ((equal (car lista) elem) (+ 1 (conta-elem elem (cdr lista))))
          (t (conta-elem elem (cdr lista)))
    )
)
;(print (conta-elem 3 '(1 2 3 3)))

(defun apaga-elem (elem lista)
    (cond
        ((null lista) NIL)
        ((equal (car lista) elem) (apaga-elem elem (cdr lista)))
        (t (cons (car lista) (apaga-elem elem (cdr lista))))
    )
)
;(print (apaga-elem 3 '(1 2 3 4 3)))

(defun monta-pares (lista)
    (cond
        ((null lista) NIL)
        (t (cons (list (car lista) (conta-elem (car lista) lista)) (monta-pares (apaga-elem (car lista) (cdr lista)))))
    )
)
;(print (monta-pares '(1 2 3 3)))

(defun predicado-principal(lista)
        (monta-pares(desparentize lista)))
;(print (predicado-principal '(a b z x 4.6 (a x) () (5 z x) () )))
```

Figura 1 - Código do exercício 01.

Fonte: Repositório no GitHub.

A função "desparentize" é responsável por transformar uma lista que pode ter vários níveis em uma lista de um único nível. Ela recebe uma lista L e primeiro verifica se esta lista é vazia; se sim, retorna "nil". Logo após, verifica se o argumento L é um átomo. Se sim, ele cria uma lista vazia e adiciona L a ela.

Em seguida, utiliza-se o conectivo lógico "*and*" para verificar se L é uma lista — vazia ou não — e se sua cabeça é nula — ou seja, se a lista é vazia. Se for, escreve-se "nil" na cabeça da lista e chama novamente a função "desparentize" na cauda de L.

Por fim, se nenhuma das verificações anteriores foi positiva, chama-se o "desparentize" na cabeça de L, que é uma lista, bem como na cauda e realiza-se o "*append*" dessas duas partes.

A função "conta-elem" recebe uma lista lista e um elemento elem e conta quantas vezes esse elemento aparece na lista.
A função verifica se lista é vazia e, se for, retorna 0.
Depois, verifica se elem encontra-se na cabeça da lista utilizando a função "*equal*". Se sim, soma-se 1 ao resultado de "conta-elem" na cauda da lista ─ juntamente com o elemento passado inicialmente.
Se o elemento não estiver na cabeça da lista, estará em sua cauda. Assim, chama-se a função passando como parâmetros o elemento elem e a cauda da lista.

A função "apaga-elem" recebe um elemento elem e uma lista lista e apaga este elemento da lista. Se a lista passada for vazia, retorna "NIL".
Novamente, utiliza-se a função "*equal*" para verificar se elem é a cabeça de lista. Se sim, chama-se "apaga-elem" na cauda de lista.
Senão, adiciona-se a cabeça de lista ao resultado da chamada de "apaga-elem" na cauda de lista.

A função "monta-pares" é responsável por criar pares contendo os elementos e o número de vezes que aparecem em uma lista. Ela recebe apenas uma lista lista e verifica se esta é vazia ─ se for, retorna "NIL".
Caso contrário: adiciona a lista formada pelo par cabeça da lista e pelo resultado da execução do "conta-elem" com "elem" sendo a cabeça de lista à outra lista formada pelo "monta-pares" na cauda de lista depois de todas as ocorrências de car(lista) terem sidos apagados.

Por fim, a função "predicado-principal" recebe uma lista lista qualquer e chama a função "monta-pares" no resultado da aplicação de "desparentize" recebendo lista.

Exemplos de execução:

Exemplo 1:

```
[1]>
DESPARENTIZE
[2]>
CONTA-ELEM
[3]>
APAGA-ELEM
[4]>
MONTA-PARES
[5]>
PREDICADO-PRINCIPAL
[6]> (predicado-principal '(a b z x 4.6 (a x) () (5 z x) ()))

((A 2) (B 1) (Z 2) (X 3) (4.6 1) (NIL 2) (5 1))
```

Exemplo 2:

```
[6]> (predicado-principal '( 1 2 (1 (2) 3) x () 1 3.5 ((x)) 1))

((1 4) (2 2) (3 1) (X 2) (NIL 1) (3.5 1))
```

2. Exercício 02:

Figura 2 - Código do exercício 02.

```
1   (defun Conta_Seguidos (X L)
2       (cond ((null L) nil)
3     ((null (cdr L)) 1)
4       ((equal X (car L))
5         (+ 1 (Conta_Seguidos X (cdr L))))
6         (t 0)))
7
8   (defun Remover_Seguidos (X L)
9   (cond ((null L) nil)
10  ((equal X (car L))
11      (Remover_Seguidos X (cdr L)))
12      (t L)))
13
14  (defun Monta_Pares(L)
15      (cond ((null L) nil)
16      (t (cons (list (Conta_Seguidos (car L) L) (car L))
17                  (Monta_Pares (Remover_Seguidos (car L) L))))))
```

Fonte: Repositório no GitHub.

Para resolver este exercício, utilizou-se uma função auxiliar para contar os elementos seguidos chamada "Conta_Seguidos", que recebe um elemento X e uma lista L.

A primeira condição é: caso a lista seja nula, retorna-se "nil".
Verifica-se em seguida se a cauda da lista é nula e, caso seja, a contagem é de 1 elemento encontrado (no caso, a cabeça).
Por fim, se X for igual à cabeça da lista L, soma-se 1 ao resultado da recursão de "Conta_Seguidos" aplicado à cauda da lista restante (cdr) em L.
Se não encontrar há elementos iguais a X, então retorna-se 0.

É realizada, então, outra função auxiliar, Remover_Seguidos**.** Tem como argumento um elemento X e uma lista L e é responsável por remover todas as outras aparições de X na lista.
Verifica se a lista é vazia; se sim, o retorno é "nil". Verifica-se se o elemento X é igual à cabeça da lista (car) L e, caso seja, chama-se recursivamente a função Remover_Seguidos passando como parâmetro o resto da lista
Se nenhuma das condições anteriores for verdadeira, retorna-se L.

Por fim, temos a função principal "Monta_Pares", que recebe uma lista L como parâmetro. Ela é responsável por montar os pares em que o primeiro componente é o número de vezes que um determinado elemento aparece na lista L e o segundo item é o próprio elemento.
É verificado se a lista é vazia. Se sim, o retorno é "nil".
Inicia-se, então, o processo de montar um par conforme mencionado anteriormente — por exemplo (4 a). Para isso, utiliza-se a função "*cons*" para adicionar a lista formada pelo retorno de "Conta_Seguidos" com os parâmetros (car L) e L à cabeça da lista. Depois, chama-se recursivamente o "Monta_Pares" para ir formando os demais pares de elementos com o resultado da aplicação de "Remover_Seguidos" com os parâmetros (car L) e L.

Exemplos de execução:

Exemplo 1:

```
[1]>
CONTA_SEGUIDOS
[2]>
REMOVER_SEGUIDOS
[3]>
MONTA_PARES
[4]> (Monta_Pares '(a a a a b c c a a d e e e e))

((4 A) (1 B) (2 C) (2 A) (1 D) (4 E))
[5]> (Monta_Pares '(a a b d c a g a d e e f f))
```

Exemplo 2:

```
[5]> (Monta_Pares '(a a b d c a g a d e e f f))

((2 A) (1 B) (1 D) (1 C) (1 A) (1 G) (1 A) (1 D) (2 E) (2 F))
```

3. Exercício 03:

Figura 3 - Código do exercício 03.

```
5   (defun lista-pares(L)
6           (cond ((null L) nil)
7           (t (append (lista-seguidos (car L)) (lista-pares (cdr L))))))
8
9   (defun lista-seguidos (L)
10          (cond ((null L) nil)
11          ((/= (car L) 0)
12              (append (cdr L) (lista-seguidos (cons (- (car L ) 1) (cdr L)))))))
```

Fonte: Repositório no GitHub.

A função "lista-seguidos" recebe uma lista L com apenas dois elementos: um número inteiro positivo e um elemento.
Ela avalia se a lista é vazia. Se for, retorna "NIL".
Senão, verifica-se a cabeça de L é diferente do número 0. Se for, faz um "*append*" entre a cauda da lista — ou seja, um elemento qualquer — e o resultado de aplicar "lista-seguidos" em uma lista composta da cabeça de L decrementado em 1 e a cauda de L.

A função "lista-pares" recebe uma lista L composta por pares de um número e uma elemento qualquer.
Ela avalia se a lista é vazia. Se for, retorna "NIL". Senão, aplica "lista-seguidos" na cabeça de L — ou seja, um par — e "lista-pares" na cauda de L — ou seja, a lista com todos os pares menos o primeiro.

Exemplo de execução:

Exemplo 1:

```
Break 15 [25]>
LISTA-SEGUIDOS
Break 15 [25]>
LISTA-PARES
Break 15 [25]> (lista-pares '((4 a) (1 b) (2 c) (2 a) (1 d) (4 e)))

(A A A A B C C A A D E E E E)
```

Exemplo 2:

```
[3]> (lista-pares '((4 a) (2 b) (8 c) (1 d) (1 d)))

(A A A A B B C C C C C C C C D D)
```