Trabalho 2 - LISP

Disciplina

Paradigmas de Linguagem de Programação - 2018-1

Grupo

Carolina Junqueira Ferreira

Julio Brito

Exercício 1

Código

```
(defun cnt (LIn)
    (conta_todos (despar LIn)))

(defun despar (lst)
  (cond ((null lst) nil)
        ((atom (car lst)) (cons (car lst) (despar (cdr lst))))
        (t (append (despar (car lst)) (despar (cdr lst))))))

(defun conta_elem (x lst)
  (let ((counter 0))
    (dolist (el lst counter) (if (equal x el) (setq counter (+ 1 counter))))
  )
)

(defun conta_todos (lst)
    (let ((pilha (elimina_rep lst)))
      (do ((aux pilha (cdr aux)) (Lout '()))
        ((null aux) Lout)
        (setq Lout (append Lout (list (list (car aux) (conta_elem (car aux)
lst)))))
      )
    )
)

(defun elimina_rep (lst)
    (let ((pilha))
        (loop for elem in lst
            do
            (if (not (member elem pilha))
                (setq pilha (append (list elem) pilha)))) pilha))
```

Descrição dos Predicados

- **cnt:** recebe uma lista como entrada para contagem;

- **despar:** recebe uma lista e a desparentiza;
  - **elimina_rep:** retorna uma lista eliminando as repetições da lista dada;
- **conta_elem:** conta todas as ocorrências de um elemento na lista dada;
- **conta_todos:** constrói a lista Lout no formato desejado no exercício. Primeiro recebe uma lista sem parênteses e cria-se uma nova lista "pilha" sem as repetições. Depois, roda-se a "pilha" e, a cada elemento, adiciona-se a Lout o par "topo/elemento da pilha" e sua contagem dentro de "lst" (lista com repetições).

Casos teste

- **Caso Exemplo:** (a b Z x 4.6 (a x) () (5 z x) ())

```
Comando: (cnt '(a b Z x 4.6 (a x) () (5 z x) ()))

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

- **Caso 1:** (MARIA (1 2 4 JOAO) (((MARIA (() joao)) MARIA b joao ())))

```
Comando: (cnt '(MARIA (1 2 4 JOAO) (((MARIA (() joao)) MARIA b joao ()))))

Saída: ((B 1) (NIL 2) (JOAO 3) (4 1) (2 1) (1 1) (MARIA 3))
```

- **Caso 2**: (MARIA (1 2 4 JOAO) (((MARIA (() joao)) MARIA b joao ())))

```
Comando: (cnt '(X1 b (1 a 2 4 JOAO) 4.3222 (((2 MARIA (() (a 4.322 ()))))) MARIA b
joao ()))))

Saída: ((4.322 1) (NIL 3) (MARIA 2) (4.3222 1) (JOAO 2) (4 1) (2 2) (A 2) (1 1) (B
2) (X1 1))
```

Exercício 2

Código

```
(defun cnt (LIn) (if (null LIn) nil (conta_todos (car LIn) 1 (cdr LIn))))

(defun conta_todos (topo n lst)
    (if (null lst)
        (list (list n topo))
        (let ((prox (car lst)))
            (if (equal prox topo)
                (conta_todos topo (+ n 1) (cdr lst))
                (cons (list n topo)
                    (conta_todos prox 1 (cdr lst)))))))
```

Descrição dos Predicados

- **cnt:** recebe uma lista como entrada para contagem. Se a lista for vazia, retorna "nil", se não, passa o topo da lista, a contagem inicial 1 e a cauda da lista para a função "conta_todos";
- **conta_todos:** é uma função recursiva que recebe os parâmetros e retorna a lista com o formato desejado no exercício. Primeiro, essa função verifica se o parâmetro "lst" é vazio, se sim, retorna o par "contagem e topo". Se a verificação anterior for falsa, obtém-se o topo da lista em "prox". Se o topo da lista atual é igual ao próximo elemento, continua a contagem com "conta_todos", passando como parâmetro o topo, contagem atual somado de 1 e a cauda da lista. Se o topo é diferente do próximo concatena-se o par "contador 'n' e topo da lista" com a lista resultante de "conta_todos" (que recebe a cauda da lista da iteração atual).

Casos teste

- **Caso Exemplo:** (a a a a b c c a a d e e e e)

```
Comando: (cnt '(a a a a b c c a a d e e e e))

Saída: ((4 A) (1 B) (2 C) (2 A) (1 D) (4 E))
```

- **Caso 1:** (a a (JOANA JOANA) () d (1 2) e e (1 2) () ((JOANA JOANA)) d e e e)

```
Comando: (cnt '(a a (JOANA JOANA) () d (1 2) e e (1 2) () ((JOANA JOANA)) d e e e))

Saída: ((2 A) (1 (JOANA JOANA)) (1 NIL) (1 D) (1 (1 2)) (2 E) (1 (1 2)) (1 NIL)
 (1 ((JOANA JOANA))) (1 D) (3 E))
```

- **Caso 2:** ((((() (() b)))) 4.32 4.32 4.322 4.3222 (b c) c JOAO JOAO (d e) e e e)

```
Comando: (cnt '((((() (() b)))) 4.32 4.32 4.322 4.3222 (b c) c JOAO JOAO (d e) e e
e))

Saída: ((1 (((NIL (NIL B))))) (2 4.32) (1 4.322) (1 4.3222) (1 (B C)) (1 C) (2 JOAO)
 (1 (D E)) (3 E))
```

Exercício 3

Código

```
(defun monta_lista (LIn)
    (if (null LIn) nil
        (let ((el (car LIn))
            (Lout (monta_lista (cdr LIn))))
```

```
            (append (apply #'pega_elem el) Lout))))

(defun pega_elem (n el)
    (if (equal n 0) nil
        (cons el (pega_elem (- n 1) el))))
```

Descrição de Predicados

- **pega_elem:** para cada quantidade "n" e elemento "el", repete o elemento "n" vezes (de n a 0);
- **monta_lista:** recursivamente, gera uma lista no formato desejado no exercício. Primeiro verifica se LIn é vazia, se verdadeiro retorna nil. Se LIn não é vazia, pega o topo da lista (par "quantidade de repetições e elemento") e atribui o valor inicial de Lout como sendo a lista resultante de "monta_lista" aplicado sobre a cauda da LIn. Por fim, aplica no par do topo de LIn a função pega_elem, e concatena a lista retornada com Lout.

Casos teste

- **Caso Exemplo:** ((4 a) (1 b) (2 c) (2 a) (1 d) (4 e))

```
Comando: (monta_lista '((4 a) (1 b) (2 c) (2 a) (1 d) (4 e)))

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

- **Caso 1:** ((2 ()) (1 A) (3 JOAO) (1 Z) (1 X) (3 4.6) (1 ()) (1 (A X)) (2 (5 () Z X)) (1 ()))

```
Comando: (monta_lista '((2 ()) (1 A) (3 JOAO) (1 Z) (1 X) (3 4.6) (1 ()) (1 (A X))
(2 (5 () Z X)) (1 ())))

Saída: (NIL NIL A JOAO JOAO JOAO Z X 4.6 4.6 4.6 NIL (A X) (5 NIL Z X) (5 NIL Z X)
NIL)
```

- **Caso 2:** ((1 ((() (() B))))) (4 999) (1 A) (1 MARIA) (2 (((()))))) (1 ()) (2 C) (2 A) (1 3) (4 1))

```
Comando: (monta_lista '((1 ((((() (() B))))) (4 999) (1 A) (1 MARIA) (2 (((()))))) (1
()) (2 C) (2 A) (1 3) (4 1)))

Saída: ((((NIL (NIL B)))) 999 999 999 999 A MARIA (((NIL))) (((NIL))) NIL C C A A 3
1 1 1 1)
```