UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA EM LISP PARA CONTAGEM DE ÁTOMOS EM LISTAS E ORDENAÇÃO

Discentes:

Caio L. R. S. Ueno
Gabriel C. P. Mendes

Docente:

Dra. Heloisa de Arruda Camargo

Ciência da Computação

Disciplina: Paradigmas de Linguagem de Programação

São Carlos / SP

6 de janeiro de 2021

1 Código-fonte

Para a realização do projeto, programou-se dois códigos: a Figura 1 ilustra as funções para contar átomos em uma lista; a Figura 2 ilustra as funções para a ordenação de listas numéricas. Ambos códigos-fonte estão disponíveis no Github.

Figura 1 – Código fonte conta átomos em Lisp.

```
(defun removeSubList(L)
    (cond((null L) ())
        ((and (listp (car L)) (not (null (car L)))) (append (removeSubList (car L)) (removeSubList (cdr L))))
        ((atom (car L)) (cons (car L) (removeSubList (cdr L))))
    )
)

(defun belongsTo(E L)
    (cond  ((null L) nil)
           ((equal (car L) E) t)
           (t (belongsTo E (cdr L)))
    )
)

(defun removeAll(E L)
    (cond  ((null L) ())
           ((equal (car L) E) (removeAll E (cdr L)))
           (t (cons (car L) (removeAll E (cdr L))))
    )
)

(defun removeDuplicates(L)
    (cond ((null L) ())
          ((belongsTo (car L) (cdr L)) (cons (car L) (removeDuplicates (removeAll (car L) L))))
          (t (cons (car L) (removeDuplicates (cdr L))))
    )
)

(defun countElem(E L)
    (cond ((null L) 0)
          ((equal (car L) E) (+ (countElem E (cdr L)) 1))
          (t (countElem E (cdr L)))
    )
)

(defun countAll(L_Orig L_Uniq)
    (cond ((null L_Uniq) ())
          (t (cons (list (car L_Uniq) (countElem (car L_Uniq) L_Orig)) (countAll L_Orig (cdr L_Uniq))))
    )
)

(defun main(L)
    (pprint (countAll (removeSubList L) (removeDuplicates (removeSubList L))))
)
```

Fonte – Criada pelos autores.

Figura 2 – Código fonte ordenação numérica em Lisp.

```
(defun menorQue(E L)
    (cond ((null L) ())
          ( (< (car L) E) (cons (car L) (menorQue E (cdr L))) )
          (t (menorQue E (cdr L)))
    )
)

(defun maiorQue(E L)
    (cond ((null L) ())
          ( (>= (car L) E) (cons (car L) (maiorQue E (cdr L))) )
          (t (maiorQue E (cdr L)))
    )
)

(defun ordenaRapido(L)
    (cond ((<= (length L) 1) L)
          (t (append (ordenaRapido (menorQue (car L) (cdr L))) (cons (car L) (ordenaRapido (maiorQue (car L) (cdr L))))) )
    )
)
```

Fonte – Criada pelos autores.

2 Explicação das funções definidas

2.1 Contagem de átomos

2.1.1 removeSubList

Função cujo resultado consiste na eliminação de listas internas, ou seja, elementos pertencentes às listas internas são colocados no primeiro nível, listas vazias são consideradas como um elemento. Primeiramente, é verificado se a lista passada está vazia, caso esteja, seu resultado é uma lista vazia. Caso contrário, segundamente, a cabeça da lista fornecida é verificada, caso seja uma lista e não seja uma lista vazia, tanto a cabeça quanto a cauda são passadas –separadamente– para a mesma função (chamada recursiva) e, após, as listas resultantes são concatenadas. Por fim, caso o elemento da cabeça da lista não seja uma lista, ele é inserido na cabeça da lista resultante e a cauda é passada para a mesma função (chamada recursiva).

2.1.2 belongsTo

Função que recebe dois argumentos - Elemento e Lista - para verificar se o primeiro pertence ao segundo. Para tal, verifica se E é a cabeça da lista - se for, então ele pertence -, caso contrário, verifica se E pertence a cauda da lista. Importante ressaltar a condição de parada: caso o segundo argumento - a lista - seja vazia, então retona-se nil (*False*).

2.1.3 removeAll

Função para remover todas as ocorrências de um dado elemento em uma dada lista. Para tal, primeiro, é verificado se a lista está vazia, caso esteja, a lista resultante é vazia. Segundo, caso a lista não seja vazia, é verificado se a cabeça da lista é o elemento que busca-se remover, caso seja, há a chamada recursiva da função passando a cauda da lista. Por fim, caso o elemento não seja o mesmo da cabeça da lista, então, ele é concatenado à lista resultante e a cauda é passada na chamada recursiva da função.

2.1.4 removeDuplicates

Função para remover todas as cópias adicionais de um elemento, mantendo apenas uma ocorrência de cada elemento na lista resultante. Primeiro, verifica-se se a lista está vazia, caso esteja, a lista resultante é vazia. Segundo, caso a lista não seja vazia, é verificado se o elemento da cabeça da lista está contido na cauda dessa, caso esteja, constrói-se uma lista cuja cabeça é a mesma da lista passada e a cauda é o resultado da chamada recursiva

dessa função passando como parâmetro a lista anterior sem a ocorrência do elemento da cabeça. Por fim, caso o elemento (a cabeça) não se repita na lista, constrói-se uma lista cuja cabeça é a mesma da lista passada e a cauda é o resultado da chamada recursiva dessa função passando como parâmetro a cauda da lista.

2.1.5 countElem

Função para contar as ocorrências de um elemento em uma lista - E e L, respectivamente. Caso a cabeça da lista seja igual o elemento desejado, então soma-se um a chamada recursiva na cauda da lista. Caso contrário, somente retorna-se a chamada recursiva na cauda. Se a lista for vazia - condição de parada - o retorno é igual a zero.

2.1.6 countAll

Função que conta as ocorrências, na primeira lista, de todos os elementos da segunda. Retorna uma lista que contém sublistas, essas são formadas pelos elementos únicos da primeira lista e suas respectivas ocorrências. Define-se a condição de parada quando a segunda lista for vazia - a lista que contém elementos únicos a serem contados.

2.1.7 main

Função que contém a lógica principal do exercício. Utiliza as funções definidas previamente para contar as ocorrências dos elementos em uma lista L dada. Imprime na tela o resultado final.

2.1.8 Exemplos

A seguir, a Figura 3 ilustra três exemplos do funcionamento da função *main*.

Figura 3 – Exemplos quanto ao Funcionamento do Programa em LISP.

```
[16]> (main '(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))
[17]> (main '(1 2 ab 3 () ((1 2 3 a))))
((1 2) (2 2) (AB 1) (3 2) (NIL 1) (A 1))
[18]> (main '((() () () ()) ((()))))
((NIL 5))
```

Fonte – Criada pelos autores.

2.2 Ordenação (*QuickSort*)

2.2.1 menorQue

Função que recebe dois argumentos, um elemento e uma lista. Retorna uma lista que contém todos os elementos da segunda que sejam menores que o primeiro. Se a lista for vazia, retorna-se uma lista vazia também.

2.2.2 maiorQue

Função similar a anterior, porém com o comparador oposto. Retorna uma lista que contém todos os elementos da segunda que sejam maiores - ou iguais - que o primeiro. Se a lista for vazia, retorna-se uma lista vazia também.

2.2.3 ordenaRapido

Função que aplica a lógica do *QuickSort*. Dado um pivô - nesse caso o primeiro elemento da lista - ordena duas listas, uma que contém os elementos menores e outra que contém os elementos maiores que o pivô, recursivamente. Após as ordenações, concatena a primeira com a segunda lista, porém com o pivô entre elas.

2.2.4 Exemplos

A Figura 4 ilustra três exemplos do funcionamento da ordenação.

Figura 4 – Exemplos quanto ao Funcionamento do ordenaRapido em LISP.

```
[22]> (ordenarapido '(4 3 2 10 88 69 100))
(2 3 4 10 69 88 100)
[23]> (ordenarapido '(10 9 8 7 6 5 4 3 2 1))
(1 2 3 4 5 6 7 8 9 10)
[24]> (ordenarapido '(5 5 10 5 5 10 33))
(5 5 5 5 10 10 33)
```

Fonte – Criada pelos autores.