Trabalho de João Pedro Pereira

Alunos:

A1 - Marcelo Riceto Bertier RA:743575

A2 - João Pedro Pereira RA:769714

Exercício 1 - A1 e A2 (Dupla)

Código:

```
(defun substitui (elemento1 elemento2 lista)
(cond ((null lista) nil)
((equal elemento1 (car lista))
(cons elemento2 (substitui elemento1 elemento2
(cdr lista))))
(t (cons (car lista) (substitui elemento1 elemento2
(cdr lista)))))
)

(defun squash (lista)
(cond ((null lista) nil)
((atom lista) (list lista))
(t (append (squash (car lista)) (squash (cdr lista)))))
)

(defun apaga (elemento lista)
(cond ((null lista) nil)
((equal elemento (car lista))
(apaga elemento (cdr lista)))
(t (cons (car lista) (apaga elemento (cdr lista)))))
)

(defun count_list (lista)
        (cond ((null lista) nil)(
        (write(cons(car lista)(count (car lista) lista)))
        (count_list (apaga (car lista) lista)))
))


(count_list (squash (substitui NIL "()" `(a b z x 4.6 (a x) () (5 z x) ())))))
```

Resumo: Dado 1 lista L, substituímos as listas vazias “()” pela string “()” utilizando a função substitui, então colocamos todas as sublistas em 1 uma lista utilizando o squash (removemos todos os “()”), e por fim utilizamos a função count_list que conta

1 elemento na lista e apaga todas as ocorrências do mesmo na lista, exibindo o elemento e a quantidade de vezes que este aparece.
Função substitui : Substitui todos as sub listas vazias pela string “()”
Função apaga : Apaga todas as ocorrências de 1 elemento da lista
Função squash : Retira todos os subníveis da lista, deixando todos os elementos e 1 lista
Função count_list: Conta quantos elementos tem de cada um na lista.

Execução do código – exemplos:

Exemplo 1:

```
[4]> (defun count_list (lista)
    (cond ((null lista) nil)(
    (write(cons(car lista)(count (car lista) lista)))
    (count_list (apaga (car lista) lista)))
))
COUNT_LIST
[5]> (count_list (squash (substitui NIL "()" `(a b z x 4.6 (a x) () (5 z x) ())))))
(A . 2)(B . 1)(Z . 2)(X . 3)(4.6 . 1)("()" . 2)(5 . 1)
```

Lista dada: (a b z x 4.6 (a x) () (5 z x) ())
Lista construída: (A . 2)(B . 1)(Z . 2)(X . 3)(4.6 . 1)("()" . 2)(5 . 1)

Exemplo 2:

```
Break 4 [10]> (count_list (squash (substitui NIL "()" `(1 25 32 64 a abelha (aviao x 1)))))
(1 . 2)(25 . 1)(32 . 1)(64 . 1)(A . 1)(ABELHA . 1)(AVIAO . 1)(X . 1)
```

Lista dada: (1 25 32 64 a abelha (aviao x 1)
Lista construída: (1 . 2)(25 . 1)(32 . 1)(64 . 1)(A . 1)(ABELHA . 1)(AVIAO . 1)(X . 1)

Exercício 2 - A2 (Individual)

Dada uma lista L com elementos numéricos, ordenar a lista em ordem crescente. Se um número aparecer mais de uma vez, as repetições devem ser mantidas no resultado.

Para ordenar a lista, usaremos o *selection* sort (Ordenação por seleção). Esse é um clássico e simples algoritmo de ordenação, sua ideia principal é mover o menor valor do vetor para a primeira posição do vetor (nesse caso uma lista).

Ademais, o processo se repete para o segundo menor valor, ficando na segunda posição do vetor (lista) e dessa maneira o processo é repetido sucessivamente até o último elemento.

Código em Lisp:

```
(defun Ordena_Lista (lista)
        (when lista
           (let ((Menor_Valor (reduce #'min lista)))
                 (cons Menor_Valor (Ordena_Lista (remove Menor_Valor lista))))))
```

Explicação do passo a passo:

Linha 1: Com o uso da função pré-definida *defun*, iremos criar a função **Ordena_Lista**, que recebe como argumento uma lista;

Linha 2: Utilizamos a estrutura de controle *when*, que avalia as expressões retorna *True* somente se for verdadeiro, senão retorna **NIL**. O comportamento do comando *when* é de certa forma semelhante ao comando *while* de outras linguagens de programação

Linha 3: Com uso do operador *let* associamos à **Menor_Valor** a função *reduce* (reduz a sequência da função &chave, chave do fim, do fim do início, do fim do valor inicial => resultado) e *reduce* recebe como argumentos a função pré-definida *min* que verifica o menor valor de uma sequência e uma lista.

Linha 4: Uma nova lista é construída utilizando **Menor_Valor** (que vai possuir o menor valor da sequência) e a lista que foi passada como argumento, porém utilizando a função pré-definida *remove* que retira um valor específico de uma sequência (no caso o menor valor da lista passada como argumento).

Execução do código – exemplos:

Exemplo 1:

```
[1]> (defun Ordena_Lista (lista)
  (when lista
    (let ((Menor_Valor (reduce #'min lista)))
      (cons Menor_Valor (Ordena_Lista (remove Menor_Valor lista))))))
ORDENA_LISTA
[2]> (ordena_lista '(27 9 23 52 32 44 1 4))
(1 4 9 23 27 32 44 52)
```

Lista desordenada: (27 9 23 52 32 44 1 4)

Lista ordenada: (1 4 9 23 27 32 44 52)

Exemplo 2:

```
[3]> (ordena_lista '(232 28 16 -1 5 -69 3))
(-69 -1 3 5 16 28 232)
[4]>
```

Lista desordenada: (232 28 16 -1 5 -69 3)

Lista ordenada: (-69 -1 3 5 16 28 232)