Trabalho 2 (PLP – 2017)

Alunos: Thiago Bretas de Souza(551899) e Alexandre Regali Seleghim(551473)

O programa possui 7 funções, sendo elas:

1 – A função DESPARENTIZA, que recebe como argumento uma lista que pode ou não possuir uma ou mais sublistas e retorna uma lista com todos os elementos da lista, movendo os elementos das sublistas para a lista principal.

```
(defun desparentiza(L)
    (cond ((null L) ())
    ((atom (car L)) (append (list (car L)) (desparentiza (cdr L))))
    (t(append (desparentiza(car L)) (desparentiza(cdr L))))))
```

Exemplo:

(write(desparentiza (list a 2 5 7 (8 9))))

(a 2 5 7 8 9)

(write(desparentiza (list e 5 (c d) 7 3)))

(e 5 c d 7 3)

2 – A função CONTAELEMENTO recebe um elemento e uma lista como parâmetros e retorna quantas vezes este elemento aparece na lista dada. Caso a lista seja vazia, a função retorna 0, indicando q o elemento não aparece na lista.

```
(defun contaElemento(a L)
    (cond ((null L) 0)
        ((= a (car L)) (+ 1 (contaElemento a (cdr L))))
        (t(contaElemento a (cdr L)))))
```

Exemplo:

(write(contaElemento '1 (list 1 1 3 4 2 1 4 1)))

(write(contaElemento '2 (list 2 5 4 2 4 3 2 2 2)))

5

3 – A função PERTENCE recebe um elemento e uma lista como argumentos e verifica se este elemento pertence à lista.

```
(defun pertence(a L)
        (cond ((null L) nil)
                ((= a (car L)) t)
                (t(pertence a (cdr L)))))
```

Exemplo:

(write(pertence '2 (list 2 3 4 5 a b)))

T

(write(pertence 'c (list 2 3 4 5 a b)))

Nil

4 – A função INTERSEC recebe duas listas como argumento e retorna os elementos que estão presentes em ambas as lista. Se o elemento aparece repetido na primeira lista, ele será retornado mais de uma vez nesta função. Para eliminar essa repetição, é utilizado, posteriormente, uma outra função aliada à esta, a ELIMINA. Neste momento não é interessante remover as repetições, pois a faremos de uma vez só para todas as listas com a função ELIMINA.

```
(defun intersec (L1 L2)
    (cond ((null L1) nil)
            ((null L2) nil)
            ((pertence (car L1) L2) (cons (car L1) (intersec (cdr L1) L2)))
            (t(intersec(cdr L1) L2))))
```

Exemplo:

Para os exemplos consideres as listas L1 = (1 1 2 3 4 5) e L2 = (1 2 9 a b c)

(write(intersec L1 L2))

(1 1 2)

Agora, considere L1 = (1 2 3 4 c) e L2 = (1 1 b c d 4)

(write(intersec L1 L2))

(1 4)

5 – A função ELIMINA recebe como argumentos um elemento e uma lista e remove todas as ocorrencias deste elemento na lista dada

```
(defun elimina(e L)
        (cond ((null L) nil)
        ((= e (car L)) (elimina e (cdr L)))
        (t(cons (car L) (elimina e (cdr L))))))
```

Exemplo:

(write(elimina '2 (list 2 3 6 a d 2 v)))

(3 6 a d v)

(write(elimina '1 (list 1 f e a 3 2 1 g)))

(f e a 3 2 g)

6 – A função MONTA recebe uma lista como argumento e gera uma nova lista, composta por pares, que, por sua vez, representam um elemento da lista seguido de quantas vezes este aparece na lista. Após montar um par, a função elimina todas as ocorrências do elemento atômico na lista.

```
(defun monta(L)
    (cond ((null L) nil)
    (t(cons (list (car L) (contaElemento (car L) L)) (monta (elimina (car L) (cdr L)))))))
```

Exemplos:

(write(monta (list 1 3 2 1 2 5 6 2)))

((1 2) (3 1) (2 3) (5 1) (6 1))

(write(monta (list 3 4 4 3 2 1)))

((3 2) (4 2) (2 1) (1 1))

7 – A função LISTAFINAL recebe duas listas e retorna uma lista formada pelos pares de elementos presentes em cada uma das listas e quantas vezes cada elemento aparece em ambas as listas. Note que, o compilador não aceita uma lista que tenha mais de um nível como argumento, então a função DESPARENTIZA deve ser chamada na chamada da função.

(defun listaFinal (L1 L2)

(setq L (intersec L1 L2))

(setq LAux(intersec L2 L1))

(setq LF(append L Laux))

(write(monta LF)))

Exemplos:

(write(listaFinal (list 2 2 4 4 6 5) (list 2 4 5 6)))

(2 2 4 4 6 5 2 4 5 6)((2 3) (4 3) (6 2) (5 2))

Para x = (1 2 2 4 (6 2 3))

(listaFinal (desparentiza x) (list 1 4 2 3 1)))

((1 3) (2 4) (4 2) (6 1) (3 2))

Prints do trabalho:

compile lisp online

Language: Common Lisp Editor: CodeMirror

```
            (t(pertence a (cdr L))))


(defun intersec (L1 L2)
      (cond ((null L1) nil)
            ((null L2) nil)
            ((pertence (car L1) L2) (cons (car L1) (intersec (cdr L1) L2)))
            (t(intersec(cdr L1) L2))))


(defun elimina(e L)
      (cond ((null L) nil)
            ((= e (car L)) (elimina e (cdr L)))
            (t(cons (car L) (elimina e (cdr L))))))


(defun monta(L)
      (cond ((null L) nil)
            (t(cons (list (car L) (contaElemento (car L) L)) (monta (elimina (car L) (cdr L)))))))

(defun listaFinal (L1 L2)
      (setq L (intersec L1 L2))
      (setq LAux(intersec L2 L1))
      (write(append L Laux))
      (setq LF(append L Laux))
      (write(monta LF)))

(setq x (list 1 2 2 4 (list 6 6 2)))
(setq y (list 3 2 4 5 6))
(write x)
(write(desparentiza x))
(write y)
(write(listaFinal (desparentiza x) y))
```

Run it (F8) Save it [+] Show input Live cooperation Put on a wall

Absolute running time: 0.14 sec, cpu time: 0.01 sec, memory peak: 5 Mb, absolute service time: 0,15 sec

```
Hello World
(1 2 2 4 (6 6 2))(1 2 2 4 6 6 2)(3 2 4 5 6)(2 2 4 6 6 2 2 4 6)((2 4) (4 2) (6 3))((2 4) (4 2) (6 3))
```

compile lisp online

Language: Common Lisp Editor: CodeMirror

```
(defun intersec (L1 L2)
      (cond ((null L1) nil)
            ((null L2) nil)
            ((pertence (car L1) L2) (cons (car L1) (intersec (cdr L1) L2)))
            (t(intersec(cdr L1) L2))))


(defun elimina(e L)
      (cond ((null L) nil)
            ((= e (car L)) (elimina e (cdr L)))
            (t(cons (car L) (elimina e (cdr L))))))


(defun monta(L)
      (cond ((null L) nil)
            (t(cons (list (car L) (contaElemento (car L) L)) (monta (elimina (car L) (cdr L)))))))

(defun listaFinal (L1 L2)
      (setq L (intersec L1 L2))
      (setq LAux(intersec L2 L1))
      (setq LF(append L Laux))
      (write(monta LF)))

(setq x (list 1 2 2 4 (list 6 6 2)))
(setq y (list 3 2 4 5 6))
(write x)
(write(desparentiza x))
(write y)
(write(listaFinal (desparentiza x) y))
(setq x (list 2 3 5 2 (list 3 3 5)))
(setq y (list 1 3 5 2 4 3 6))
(write x)
(write(desparentiza x))
(write y)
(listaFinal (desparentiza x) y)
```

Run it (F8) Save it [+] Show input Live cooperation Put on a wall

Absolute running time: 0.14 sec, cpu time: 0.01 sec, memory peak: 5 Mb, absolute service time: 0,15 sec

```
TRABALHO T2 DE LISP
Alunos: Thiago Bretas de Souza(551899) e Alexandre Regali Seleghim(551473)
(1 2 2 4 (6 6 2))(1 2 2 4 6 6 2)(3 2 4 5 6)((2 4) (4 2) (6 3))((2 4) (4 2) (6 3))(2 3 5 2 (3 3 5))(2 3 5 2 3 3 5)(1 3 5 2 4 3 6)((2 3) (3 5) (5 3))
```

```
TRABALHO T2 DE LISP
Alunos: Thiago Bretas de Souza(551899) e Alexandre Regali Seleghim(551473)
EXEMPLO 1
n
Lista 1:
(1 2 2 4 (6 6 2))n
Lista 1 desparentizada:
(1 2 2 4 6 6 2)n
Lista 2:
(3 2 4 5 6)n
Lista Final:
((2 4) (4 2) (6 3))n
EXEMPLO 2
n
Lista 1:
(2 3 5 2 (3 3 5))n
Lista 1 desparentizada:
(2 3 5 2 3 3 5)n
Lista 2:
(1 3 5 2 4 3 6)n
Lista Final:
((2 3) (3 5) (5 3))
```

Código do programa:

```
(write-line "TRABALHO T2 DE LISP")
(write-line "Alunos: Thiago Bretas de Souza(551899) e Alexandre Regali Seleghim(551473)")


(defun desparentiza(L)
    (cond ((null L) ())
    ((atom (car L)) (append (list (car L)) (desparentiza (cdr L))))
    (t(append (desparentiza(car L)) (desparentiza(cdr L))))))


(defun contaElemento(a L)
    (cond ((null L) 0)
        ((= a (car L)) (+ 1 (contaElemento a (cdr L))))
        (t(contaElemento a (cdr L)))))

(defun pertence(a L)
    (cond ((null L) nil)
        ((= a (car L)) t)
        (t(pertence a (cdr L)))))


(defun intersec (L1 L2)
    (cond ((null L1) nil)
        ((null L2) nil)
        ((pertence (car L1) L2) (cons (car L1) (intersec (cdr L1) L2)))
        (t(intersec(cdr L1) L2))))


(defun elimina(e L)
```

```
    (cond ((null L) nil)
        ((= e (car L)) (elimina e (cdr L)))
        (t(cons (car L) (elimina e (cdr L))))))


(defun monta(L)
    (cond ((null L) nil)
        (t(cons (list (car L) (contaElemento (car L) L)) (monta (elimina (car L) (cdr L)))))))


(defun listaFinal (L1 L2)
    (setq L (intersec L1 L2))
    (setq LAux(intersec L2 L1))
    (setq LF(append L Laux))
    (write(monta LF)))
```