PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

LINGUAGENS DE PROGRAMAÇÃO

CIÊNCIA DA COMPUTAÇÃO

LUCAS OMAR ANDRADE LEAL

LISTA REVISÃO LISP

POÇOS DE CALDAS - MG

2021

RESUMO

- ***Número***
  - *nenhum número é uma variável*
  - *TODO NÚMERO É ÁTOMO*
- ***Átomo***
  - *qualquer símbolo não organizado em formato de lista*
    - *12, bola, ol42, teste, 1, 2, 3*
    - *nil é um valor nulo*
- ***Lista***
  - *conjunto de átomos ou conjunto de outras listas*
    - *(1 2 casa 4 camisa 76 arvore)*
    - *(( ))*
    - *((1 2 3) banana mamao (casa teste))*
- ***Apóstrofo (‘)***
  - *faz com que o programa reconheça a variável como átomo, e não variável*

```
> (evenp casa)
unbound variable - CASA
> (evenp ‘casa)
error: bad argument type - CASA
> (setq x 4)
> (evenp x)
T
> (setq y 5)
> (evenp y)
NIL
```

- ***Lista***
  - *sempre que tratar com lista, deve-se colocar o apóstrofo na frente para que o lisp entenda a lista como um átomo(uma coisa só)...inteira*

```
> (LIStP (a b (c d e)))
T
> (listp ‘())
T ; é uma lista vazia
```

- ***Funções***

```
> (defun f1(x y) (+ x y) )
F1
> (f1 3 4)
7
```

(01) Sejam as expressões em LISP. Escreva o retorno de cada uma delas:

a) `> (cons 1 '(b c d) ) ;coloca 1 na lista (bcd)`

```
(1 b c d)
```

b) `> (cons 1 (cons 2 nil) ) ;coloca 1 na lista (cons 2 nil)`

```
(1 2)
```

c) `> (cons '(1 2 3) (cdr '(a b c d) ) ) ;coloca a primeira lista na cauda da segunda`

```
(1 2 3 b c d)
```

d) `> (car (cdr '(a b c d) ) ) ;seleciona a cabeça da cauda da lista`

```
(b)
```

e) `> (car '( (a b c d) nil) ) ;seleciona a cabeça da lista que tem lista vazia dentro`

```
(a b c d)
```

f) `> (cdr '( (a b c d) nil ) ) ;seleciona a cauda da lista que tem a lista + a lista vazia`

```
( () )
```

g) `> (car (cdr (car (cdr '( ( ( (a b) (c d) ) (e f) ) (g h) ) ) ) ) )`

```
(car (cdr (car '(g h) ) ) )
(car (cdr ‘(g h)) )
(car ‘(h) )
(h)
```

h) `> (cons (car '(a b f) ) (cons (cons 'c '(x) ) nil) )`

```
(cons (car '(a b f) ) (cons ‘(c x) nil) )
(cons (car '(a b f) ) (c x) )
(cons (a) (c x) )
(a c x)
```

(02) Escreva a expressão abaixo em forma prefixa para que esta possa avaliada pelo interpretador LISP ((12/4) * ((3+2) / (15+1)))

```
1. (* (/ 12 4) (/ (+ 3 2) (+ 15 1)))
2. (* 3 (/ 5 16))
3. (* 3 0,3125)
4. 15/16
```

(03) Escreva uma função que receba duas listas e retorne a união delas. A função pertence abaixo deverá ser utilizada.

```
(defun pertence (elem lista)
      (cond
            ( (null lista) nil)
            ( (equal (car lista) elem) t)
            (t (pertence elem (cdr lista) ) )
      )
)
```

Ex.: a união da lista (a b c d e f) com a lista (e f g h) será (a b c d e f g h)

https://highlight.hohli.com/index.php

```
(defun pertence (elem lista)
    (cond
        ( (null lista) nil)
        ( (equal (car lista) elem) t)
        (t (pertence elem (cdr lista) ) )
    )
)

(defun merge (L1 L2)
    (cond
        ( (null L1) L2)
        ( (ItsFrom (car L1) L2) (merge (cdr L1) L2) )
        (t (cons (car L1) (merge (cdr L1) L2) ) )
    )
)
```

(04) Escreva uma função que receba uma lista e retorne a lista com o cubo de cada um de seus elementos (para isso, utilize uma lista simples (exemplo: (1 2 3 4)) e não utilize lista de lista (exemplo: (1 2 (3 4))).

Ex.: o cubo da lista (1 2 3) será (1 8 27)

```
(defun cubo (L)
(cond ( (null L) nil)
        (t (cons (* (car L)(car L) (car L) ) (cubo (cdr L) ) ) )
    )
)
```

(05) Escreva uma função que receba uma lista e retorne a lista com os elementos pares multiplicados por -1 e os ímpares removidos da lista

Ex.: para a lista (1 -2 3 4 -5 6) = (2 -4 6)

```
(defun mul_par (L)
  (cond ( (null L) nil)
    ( (evenp (car L) ) (cons (* -1 (car L) ) (mul_par (cdr L) ) ) )
    (t (mul_par (cdr L) ) )
  )
)
(print (mul_par '(1 -2 3 4 -5 6) ) )
```