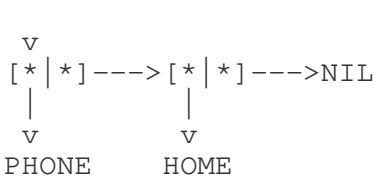
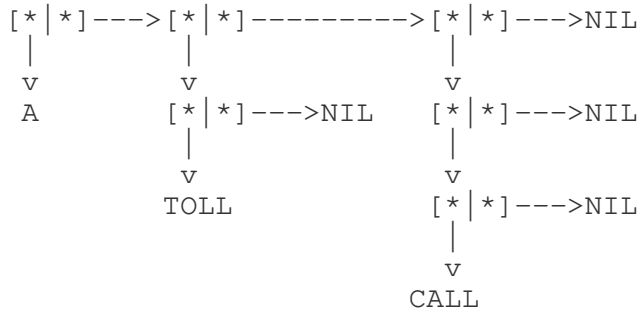
```
;;;;============================================================================;;;;
;;;; Common Lisp: A Gentle Introduction to Symbolic Computation                 ;;;;
;;;; http://www.paulgraham.com/acl.html                                         ;;;;
;;;;----------------------------------------------------------------------------;;;;
;;;; Exercícios: 2                                                              ;;;;
;;;;----------------------------------------------------------------------------;;;;
;;;; Por: Abrantes Araújo Silva Filho                                           ;;;;
;;;;      abrantesasf@pm.me                                                     ;;;;
;;;;============================================================================;;;;


;;; Exercício 2.1:
;;; --------------

[*|*]--->[*|*]--->[*|*]--->[*|*]--->[*|*]--->[*|*]--->NIL
 |        |        |        |        |        |
 v        v        v        v        v        v
 TO       BE       OR      NOT       TO       BE


;;; Exercício 2.2:
;;; --------------

; não      (A B (c)
; sim      ((A) (B))
; não      a b ) (c d)
; sim      (a (b (c)))
; sim      (((a) (b)) (c))


;;; Exercício 2.3:
;;; --------------

[*|*]--->[*|*]------------------>[*|*]--->NIL
 |        |                       |
 v        v                       v
PLEASE   [*|*]--->[*|*]--->NIL   VALENTINE
          |        |
          v        v
          BE       MY


;;; Exercício 2.4:
;;; --------------

; ((bows arrows) (flowers chocolates))


;;; Exercício 2.5:
;;; --------------

(open the pod bay doors hal) ; 6

((open) (the pod bay doors) hal) ; 3

((1 2 3) (4 5 6) (7 8 9) (10 11 12)) ; 4

((one) for all (and (two (for me)))) ; 4

((q spades)
```

```
 (7 hearts)
 (6 clubs)
 (5 diamonds)
 (2 diamonds)) ; 5

((pennsylvania (the keystone state))
 (new-jersey (the garden state))
 (massachusetts (the bay state))
 (florida (the sunshine state))
 (new-york (the epire state))
 (indiana (the hoosier state))) ; 6


;;; Exercício 2.6:
;;; --------------

() = nil

(()) = (nil)

((())) = ((nil))

(() ()) = (nil nil)

(() (())) = (nil (nil))


;;; Exercício 2.7:
;;; --------------

;; Primeio a fução REST retornará (if you like geese) e, depois,
;; a função FIRST retornará "if".


;;; Exercício 2.8:
;;; --------------

(defun my-third (lst)
  (first (rest (rest lst))))

(my-third '(a b c d))


;;; Exercício 2.9:
;;; --------------

(defun my-third (lst)
  (second (rest lst)))

(my-third '(1 2 3 4))


;;; Exercício 2.10:
;;; ---------------

[*|*]--->NIL
 |
 v
[*|*]--->NIL
 |
```




```
(car '(((phone home))))
(cdr '(((phone home))))


;;; Exercício 2.11:
;;; ---------------
```

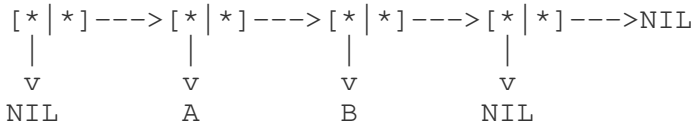



```
;;; Exercício 2.12:
;;; ---------------

(cadddr '(1 2 3 4 5))


;;; Exercício 2.13:
;;; ---------------

Para obter "fun":
C___R   (((fun)) (in the) (sun))
C__AR    ((fun))
C_AAR     (fun)
CAAAR      fun
(caaar '(((fun)) (in the) (sun)))

Para obter "in":
C___R   (((fun)) (in the) (sun))
C__DR   (        (in the) (sun))
C_ADR            (in the)
CAADR             in
(caadr '(((fun)) (in the) (sun)))

Para obter "the":
C____R   (((fun)) (in the) (sun))
C___DR   (        (in the) (sun))
C__ADR            (in the)
C_DADR               (the)
CADADR                the
(cadadr '(((fun)) (in the) (sun)))

Para obter "sun":
C____R   (((fun)) (in the) (sun))
C___DR   (        (in the) (sun))
C__DDR   (                 (sun))
```

```
C_ADDR                      (sun)
CAADDR                       sun
(caaddr '(((fun)) (in the) (sun)))


;;; Exercício 2.14:
;;; ---------------

caadr da esquerda para a direita:
C___R    ((blue cube) (red pyramid))
CA__R     (blue cube)
CAA_R      blue
CAADR      => ERRO, não é CONS <=

(cdaar '((blue cube) (red pyramid)))


;;; Exercício 2.15:
;;; ---------------

((a b) (c d) (e f))

Função      Resultado
CAR         (a b)
CDDR        ((e f))
CADR        (c d)
CDAR        (b)
CADAR       b
CDDAR       nil
CAAR        a
CDADDR      (f)
CAR CDADDR  f


;;; Exercício 2.16:
;;; ---------------

(caar '(fred nil)) => ERRO!


;;; Exercício 2.17:
;;; ---------------

(post no bills)
CAR    post
CDR    (no bills)

((post no) bills)
CAR    (post no)

(bills)
CDR    nil

bills
CAR    erro!

(post (no bills))
CDR    ((no bills))

((post no bills))
```

```
CDR    nil

nil
CAR    nil


;;; Exercício 2.18:
;;; ---------------

(defun faz-lista (x y)
  (cons x (cons y ())))

(faz-lista 'a 'b)


;;; Exercício 2.19:
;;; ---------------

(list 'fred 'and 'wilma)    =>  (fred and wilma)

(list 'fred '(and wilma))   =>  (fred (and wilma))

(cons 'fred '(and wilma))   =>  (fred and wilma)

(cons nil nil)              =>  (nil)

(list nil nil)              =>  (nil nil)


;;; Exercício 2.20:
;;; ---------------

(list nil)      =>  (nil)

(list t nil)    =>  (t nil)

(cons t nil)    =>  (t)

(cons '(t) nil) =>  ((t))

(list '(in one ear and) '(out the other))  => ((in one ear and) (out the other))

(cons '(in one ear and) '(out the other))  => ((in one ear and) out the other)


;;; Exercício 2.21:
;;; ---------------

(defun aninha2 (x y z w)
  (list (list x y) (list z w)))

(aninha2 1 2 3 4)


;;; Exercício 2.22:
;;; ---------------

(defun duo-cons (x y lst)
  (cons x (cons y lst)))
```

```
(duo-cons 'patrick 'seymour '(marvin))


;;; Exercício 2.23:
;;; ---------------

(defun two-deeper-list (x)
  (list (list x)))

(two-deeper-list 'moo)
(two-deeper-list '(bow wow))

(defun two-deeper-cons (x)
  (cons (cons x nil) nil))

(two-deeper-cons 'moo)
(two-deeper-cons '(bow wow))


;;; Exercício 2.24:
;;; ---------------

(((good)) ((night)))

(caaadr '(((good)) ((night))))


;;; Exercício 2.25:
;;; ---------------

;; Porque a função CONS contrói e retorna uma cons cell.


;;; Exercício 2.26:
;;; ---------------

(a b c)
(length (cdr '(a b c)))  => 2
(cdr (length '(a b c)))  => ERRO


;;; Exercício 2.27:
;;; ---------------

;; Quando as listas são aninhadas, pois sempre haverá mais cons cells do que o
;; número de elementos no primeiro nível.
;; Por exemplo: a lista ((a) b) tem 2 elementos, mas terá 3 cons cells:

[*|*]--------->[*|*]--->NIL
 |              |
 v              v
[*|*]--->NIL    B
 |
 v
 A

;; Já quando a lista é plana, sempre haverá o menos número de elementos
;; e o de cons cells (mesmo se os elementos forem nil).
;; Por exemplo: a lista (nil a b nil) tem 4 elementos e 4 cons cells:
```




```
;;; Exercício 2.28:
;;; ---------------

;; Se não soubermos o tamanho da lista não será possível, pois não
;; saberemos quantos vezes teremos que chamar a função CDR. Lisp
;; tem uma função que retorna uma lista com o último elemento
;; da lista original: LAST.

(last '(a b c d e f g h i j k l m))  =>  (M)
```