```
;;;; Funções para o estudo do livro "The Little Schemer", usando o dialeto
;;;; Common Lisp (implementação SBCL) ao invés do dialeto Scheme.
;;;; Abrantes Araújo Silva Filho
;;;; abrantesasf@gmail.com

;;; Chapter 01: Toys

;; Cria função "atom?" para indicar os atoms. Esta função considera que a lista
;; vazia não é um atom (diferentemente do padrão de Common Lisp).
(defun atom? (x)
  (not (listp x)))

;; Cria função "list?" apenas para padronizar a chamada:
(defun list? (x)
  (listp x))

;; Cria função "null?" apenas para padronizar a chamada.
(defun null? (x)
  (null x))

;; Cria grupo de funções de comparação para padronizar a chamada:
(defun eq? (x y)
  (eq x y))

(defun eql? (x y)
  (eql x y))

(defun equal? (x y)
  (equal x y))

(defun equalp? (x y)
  (equalp x y))




;;; Chapter 02: Again

;; Cria função "lat?" para verificar se uma lista é composta somente por atoms.
;; Atenção: esta função utiliza funções definidas anteriormente!
(defun lat? (l)
  (cond
    ((atom? l) nil)                    ; previne erro se argumento for atom
    ((null? l) t)                      ; NIL é lat!
    ((atom? (car l)) (lat? (cdr l)))   ; CAR é lat? avalia CDR recursivamente
    (t nil)))                          ; retorna falso

;; Cria função "member?' para verificar se um atom está contido em uma lat.
;; Atenção: esta função utiliza funções definidas anteriormente!
(defun member? (a lat)
  (cond
    ((null? lat) nil)
    ((or (eql? a (car lat))
         (member? a (cdr lat))))))

;; Cria função "memberl?" que generaliza a função "member?" para verificar
;; se listas estão contidas em outras listas.
(defun memberl? (s l)
  (cond
    ((null? l) nil)
    ((or (equal? s (car l))
         (memberl? s (cdr l))))))


```

```
65
66
67  ;;; Chapter 03: CONS
68
69  ;; Cria função "rember" que procura a primeira ocorrência de um atom em uma
70  ;; lista, e retorna uma nova lista sem essa primeira ocorrência.
71  (defun rember (a lat)
72    (cond
73      ((null? lat) '())
74      ((eql? a (car lat)) (cdr lat))
75      ((cons (car lat) (rember a (cdr lat))))))
76
77  ;; Cria função "remberl" que expande a função anterior para lidar com listas.
78  (defun remberl (s l)
79    (cond
80      ((null? l) '())
81      ((equal? s (car l)) (cdr l))
82      ((cons (car l) (remberl s (cdr l))))))
83
84  ;; Cria função "firsts" que retorna uma lista com todos os primeiros elementos
85  ;; de outras listas recebidas como argumento
86  (defun firsts (l)
87    (cond
88      ((null? l) '())
89      ((cons (car (car l)) (firsts (cdr l))))))
90
91  ;; Cria função "insertR" que insere um elemento novo após um elemento
92  ;; antigo em uma lat, retornando uma nova lista:
93  (defun insertR (new old lat)
94    (cond
95      ((null? lat) '())
96      ((equal? old (car lat)) (cons old (cons new (cdr lat))))
97      ((cons (car lat) (insertR new old (cdr lat))))))
98
99  ;; Cria função "insertL" que insere um elemento novo antes de um elemento antigo
100 ;; em uma lat, retornando uma nova lista:
101 (defun insertL (new old lat)
102   (cond
103     ((null? lat) '())
104     ((equal? old (car lat)) (cons new (cons old (cdr lat))))
105     ((cons (car lat) (insertL new old (cdr lat))))))
106
107 ;; Cria função "substS" que troca um elemento old por um new em uma lat,
108 ;; retornando uma nova lista. ATENÇÃO: CLisp já tem uma função "subst",
109 ;; por isso essa aqui é chamada de "substS", de "substituiu uma S-exp".
110 (defun substS (new old lat)
111   (cond
112     ((null? lat) '())
113     ((equal? old (car lat)) (cons new (cdr lat)))
114     ((cons (car lat) (substS new old (cdr lat))))))
115
116 ;; Cria função "substS2" que troca OU a primeira ocorrência de o1 OU a primeira
117 ;; ocorrência de o2 por new.
118 (defun substS2 (new o1 o2 lat)
119   (cond
120     ((null? lat) '())
121     ((or (equal? o1 (car lat))
122          (equal? o2 (car lat)))
123      (cons new (cdr lat)))
124     ((cons (car lat) (substS2 new o1 o2 (cdr lat))))))
125
126 ;; Cria a função "multirember" que remove TODAS as ocorrências de um atom de uma
127 ;; lat, retornando uma nova lista.
128 (defun multirember (a lat)
```

```
  (cond
    ((null? lat) '())
    ((equal? a (car lat)) (multirember a (cdr lat)))
    ((cons (car lat) (multirember a (cdr lat))))))

;; Cria a função "multiinsertR" que insere um novo elemento após (à direita) de
;; TODAS as ocorrências de um elemento antigo em uma lat, e retorna nova lat.
(defun multiinsertR (new old lat)
  (cond
    ((null? lat) '())
    ((equal? old (car lat)) (cons old
                                  (cons new (multiinsertR new old (cdr lat)))))
    ((cons (car lat) (multiinsertR new old (cdr lat))))))

;; Cria função "multiinsertL" que insere um novo elemento antes (à esquerda) de
;; TODAS as ocorrências de um elemento antigo em uma lat, retornando nova lat.
(defun multiinsertL (new old lat)
  (cond
    ((null? lat) '())
    ((equal? old (car lat)) (cons new
                                  (cons old (multiinsertL new old (cdr lat)))))
    ((cons (car lat) (multiinsertL new old (cdr lat))))))

;; Cria função "multisubst" que troca TODAS as ocorrências de um elemento antigo
;; em uma lat por um elemento novo, retornando uma nova lat.
(defun multisubst (new old lat)
  (cond
    ((null? lat) '())
    ((equal? old (car lat)) (cons new (multisubst new old (cdr lat))))
    ((cons (car lat) (multisubst new old (cdr lat))))))

;; Cria função "multisubst2" que troca TODAS as ocorrência de um elemento antigo
;; OU outro elemento antigo em uma lat, por um elemento novo, retorna nova lat.
(defun multisubst2 (new o1 o2 l)
  (cond
    ((null? l) '())
    ((or (equal? o1 (car l))
         (equal? o2 (car l)))
     (cons new (multisubst2 new o1 o2 (cdr l))))
    ((cons (car l) (multisubst2 new o1 o2 (cdr l))))))
end
```