```lisp
;============================== Lab 7 (1.2) ====================================
; tests:
; (set-equal '(1 2 3) '(1 2 3 3 3)) ; t
; (set-equal '() '(12))
; (set-equal '(1 2 3) '(1 2))

(defun my-memberp (elem lst)
  (and lst
       (equal elem (car lst))
       (my-memberp elem (cdr lst))))

(defun my-subset (lst1 lst2)
  (cond ((null lst1) T)
        ((my-member (car lst1) lst2) (my-subset (cdr lst1) lst2))
        (T Nil)))

(defun set-equal (lst1 lst2)
  (and (my-subset lst1 lst2) (my-subset lst2 lst1)))

;============================== Lab 7 (1.4) ====================================

;swap-first-last without reverse

(defun change-last-cons (lst change-on)
  (cond ((null lst) lst)
        ((null (cdr lst)) (cons change-on nil))
        (T (cons (car lst) (change-last-cons (cdr lst) change-on)))))
(defun swap-first-last-cons (lst)
  (change-last-cons (cons (car (my-last lst)) (cdr lst)) (car lst)))

;============================== Lab 8 (1.4) ====================================

;helpers:
(defun flat-map (mapper lst)
  (mapcan #'(lambda (x)
              (cond ((atom x) (list (funcall mapper x)))
                    (T (square-lst x))))
          lst))

(defun flat-map-rec-internal (mapper lst acc)
  (cond
    ((null lst) acc)
    ((atom (car lst)) (flat-map-rec-internal mapper (cdr lst) (cons (funcall mapper (car lst)) acc)))
    (T (flat-map-rec-internal mapper (cdr lst) (flat-map-rec-internal mapper (car lst) acc)))))

(defun flat-map-rec (mapper lst)
```

```
  (reverse (flat-map-rec-internal mapper lst Nil)))

;helpers end.

; recursions

(defun lst-minus-10-rec-flat (lst)
  (flat-map-rec
    #'(lambda (x)
        (cond ((numberp x) (- x 10))
              (T x)))
    lst))

(defun lst-minus-10-rec (lst)
  (and lst
       (cond ((numberp lst) (- lst 10))
             ((symbolp lst) lst)
             (T (cons (lst-minus-10-rec (car lst)) (lst-minus-10-rec (cdr lst)))))))

; functionals

(defun lst-minus-10-func (lst)
  (mapcar #'(lambda (elem)
              (cond ((numberp elem) (- elem 10))
                    ((atom elem) elem)
                    (T (cons (lst-minus-10-func (car elem)) (lst-minus-10-func (cdr
                        elem))))))
          lst))

(defun lst-minus-10-func-flat (lst)
  (flat-map
    #'(lambda (x)
        (cond ((numberp x) (- x 10))
              (T x)))
    lst))

;============================== Lab 9 (1.1.2) ===================================

;helpers:

(defun flat-filter-map-rec-internal (filter-mapper lst acc)
  (cond
    ((null lst) acc)
    ((atom (car lst)) (flat-filter-map-rec-internal filter-mapper (cdr lst)
                                                    (funcall filter-mapper (car lst) acc)))
    (T (flat-filter-map-rec-internal filter-mapper (cdr lst)
                                     (flat-filter-map-rec-internal filter-mapper (car lst)
                                         acc)))))
```

```
93
94  (defun flat-filter-map-rec (filter-mapper lst)
95    (reverse (flat-filter-map-rec-internal filter-mapper lst Nil)))
96
97  (defun my-last (lst)
98    (cond ((null lst) lst)
99          ((null (cdr lst)) lst)
100         (T (my-last (cdr lst)))))
101 (defun my-nconc (lst elem)
102   (cond ((null lst) elem)
103         (T (setf (cdr (my-last lst)) elem)
104            lst)))
105
106 (defun my-mapcan-internal (mapper lst acc)
107   (cond ((null lst) acc)
108         (T (let ((val (funcall mapper (car lst))))
109              (cond ((listp val) (my-mapcan-internal mapper (cdr lst) (my-nconc acc val)))
110                    (T (my-mapcan-internal mapper (cdr lst) acc)))))))
111 (defun my-mapcan (mapper lst)
112   (my-mapcan-internal mapper lst nil))
113
114 (defun filter-map-rec (filter-mapper lst)
115   (my-mapcan #'(lambda (x)
116                  (cond ((listp x) (let ((res (filter-map-rec filter-mapper x)))
117                                     (and (consp res) (list res))))
118                        (T (funcall filter-mapper x))))
119              lst))
120 ;helpers end.
121
122 (defun list-btwn-only-flat (from to lst)
123   (flat-filter-map-rec
124     #'(lambda (x acc)
125         (cond ((and (numberp x) (< x to) (> x from)) (cons x acc))
126               (T acc)))
127     lst))
128
129 (defun list-btwn-only (from to lst)
130   (filter-map-rec #'(lambda (x)
131                       (and (numberp x) (< x to) (> x from) (list x)))
132                   lst))
133
134 ;* (list-btwn-only 0 6 '(((a)) b c))
135 ;NIL
136 ;* (list-btwn-only 0 6 '(a b 1 2 3 6))
137 ;(1 2 3)
138 ;* (list-btwn-only 0 6 '(1 (2 3 4 (5 6))))
139 ;(1 (2 3 4 (5)))
140
```

```
;============================== Lab 10 (1.5.2) ====================================

;helpers
(defun my-nthcdr (n lst)
  (and lst
       (cond ((<= n 0) lst)
             (T (my-nthcdr (- n 1) (cdr lst))))))
;helpers end.

(defun every-md (m d lst func default-val)
  (cond ((or (null lst) (< m 0)) default-val)
        (T (funcall func (car lst) (every-md (- m d) d (my-nthcdr d lst) func
            default-val)))))

(defun every-nmd (n m d lst func default-val)
  (every-md (- m n) d (nthcdr n lst) func default-val))

(defun sum-elem (elem)
  (cond ((numberp elem) elem)
        ((symbolp elem) 0)
        (T (sum-pair (car elem) (cdr elem)))))
(defun sum-pair (el1 el2)
  (+ (sum-elem el1) (sum-elem el2)))

(defun sum-nmd (n m d lst)
  (every-nmd n m d lst #'sum-pair 0))

;(sum-nmd 1 8 2 '(0 (1 2 3 4) 2 (1 (2 (3 (4)))) 4 (1 a 2 b 3 c 4 d) 6 (1 ((a)) (((2))) 3
    4) 8 9 10 11))


;============================================================================

;optional helpers:
(defun my-append-cons (lst elem)
  (cond ((null lst) lst)
        ((null (cdr lst)) (cons (car lst) elem))
        (T (cons (car lst) (my-append-cons (cdr lst) elem)))))

(defun move (from to)
  (cond ((null from) to)
        (T (move (cdr from) (cons (car from) to)))))
(defun my-append-internal (lst elem acc)
  (cond ((null lst) (move elem acc))
        (T (my-append-internal (cdr lst) elem (cons (car lst) acc)))))
(defun my-append (lst elem)
  (reverse (my-append-internal lst elem nil)))
;optional helpers end.
```