Partial Solution to Assignment-1

sol. to Q-1 (part b)

- --- part B.1 ---
  - fully dotted notation
    ((((C . nil) . B) . nil) . ((D . nil) . ((E . F) . ((nil . nil) .nil))))
  - simplified
    ((((C) . B)) (D) (E . F) (nil))
- --- part B.2 ---
  - fully dotted notation
    ((A . (nil . (nil . B))) . (C . ((E . (F . nil)) . D)))
  - simplified
    ((A nil nil . B) C (E F) . D)

sol. to Q-2

- (caadadr `(a `(b (c))))
  = (caadadr (quote (a (quote (b (c))))))
- after evaluating the argument, the actual argument supplied to caadadr is
  (a (quote (b (c))))
- caadadr (a (quote (b (c)))) =
  caadar ((quote (b (c)))) =
  caadr (quote (b (c))) =
  caar ((b (c))) =
  car (b (c)) =
  b

sol. to Q-3

- static: 16
- dynamic: 18

sol. to Q-4.1

```
(defun func41 (item aList)
   (cond ((equal item (caar aList))
                (second (car aList)))
         (t  (func41 item (cdr aList)))
   )
)
```


sol. to Q-4.2

```
(defun func42 (wff aList)
    (cond ((equal t wff) t)
          ((equal nil wff) nil)
          ((atom wff) (func41 wff Alist))
          ((equal 'null (car wff)) (null (func42 (cadr wff) aList)))
          ((equal 'and (car wff)) (and   (func42 (cadr wff) aList)
                                         (func42 (caddr wff) aList)))
          ((equal 'or (car wff)) (or     (func42 (cadr wff) aList)
                                         (func42 (caddr wff) aList)))
      )
)
```

sol. to Q-5

- (defun row-sum (matrix)
  (mapcar (bu 'reduce '+) matrix))
- (defun column-sum (matrix)
  (row-sum (trans matrix)))
- (defun matrix-sum (m1 m2)
  (mapcar 'column-sum (trans (list m1 m2)))
)

sol. to Q-6.1, 6.2

- (defun map2nd-level (func list)
  (mapcar (bu 'mapcar func) list))
- (defun map2op-v1 (f g list)
  (mapcar (comp f g) list))
- (defun map1op-v2 (f-g-list)
  (mapcar (comp (first f-g-list)
  (second f-g-list))
  (third f-g-list)))

sol. to Q-6.3

- (defun diagonals (n)
  (apply 'append (mapcar 'gen-the-list (range 0 n)))
)
- (defun gen-the-list (i)
  (genlist (1+ i) 'next-tuple (list 0 i))
)
- (defun next-tuple (tuple)
  (list (1+ (car tuple)) (1- (cadr tuple))))