

13.

- a) A linear list is given. Eliminate from the list all elements from N to N steps, N-given.
- b) Write a function to test if a linear list of integer numbers has a "valley" aspect (a list has a valley aspect if the items decrease to a certain point and then increase. Eg. 10 8 6 17 19 20). A list must have at least 3 elements to fulfill this condition.
- c) Build a function that returns the minimum numeric atom from a list, at any level.
- d) Write a function that deletes from a linear list of all occurrences of the maximum element.

a)

```
(defun remove-every-n (lst n)
  (labels ((helper (l count)
    (cond
      ((null l) nil)
      ((= count n)
       (helper (cdr l) 1)))
      (t
       (cons (car l)
         (helper (cdr l) (+ count 1)))))))
  (helper lst 1)))
```

```
(print (remove-every-n '(1 2 3 4 5 6 7 8 9) 3))
```

```
-> (1 2 4 5 7 8)
```

$L = \langle x_1, x_2, x_3, \dots, x_n \rangle$

N = given step.

R([], n, i) = []

R(x :: xs, n, i) =

if i = n

then R(xs, n, 1)

else x :: R(xs, n, i+1)

removeEveryN(L, n) = R(L, n, 1)

b)

```
(defun valley-p (lst)
```

```
  (if (< (length lst) 3)
```

```
    nil
```

```
    (let ((down t))
```

```
(labels ((check (l)
  (cond
    ((null (cdr l)) t)
    ((and down (> (car l) (cadr l)))
     (check (cdr l)))
    ((and down (< (car l) (cadr l)))
     (setf down nil)
     (check (cdr l)))
    ((and (not down) (< (car l) (cadr l)))
     (check (cdr l)))
    (t nil))))
  (check lst))))
```

(print (valley-p '(10 8 6 17 19 20)))

-> T

Let L = $\langle x_1, x_2, \dots, x_k \rangle$, k ≥ 3.

V([a,b] , decreasing) = true if decreasing and a > b

V([a,b] , increasing) = true if (not decreasing) and a < b

c)

```
(defun min-atom (lst)
  (labels ((find-min (l current-min)
    (cond
      ((null l) current-min)
      ((numberp (car l))
       (if (< (car l) current-min)
           (find-min (cdr l) (car l))
           (find-min (cdr l) current-min)))
      (t
       (let ((sub-min (find-min (car l) current-min)))
         (find-min (cdr l) sub-min)))))))
  (find-min lst 999999999)))
```

(print (min-atom '(7 (3) (9 (2 10)))))

-> 2

Min(atom) = atom
Min([]) = +∞
Min(x :: xs) = min(Min(x), Min(xs))

d)

```
(defun max-atom (lst)
  (labels ((find-max (l current-max)
    (cond
      ((null l) current-max)
      ((> (car l) current-max)
       (find-max (cdr l) (car l)))
      (t
       (find-max (cdr l) current-max))))))
  (find-max lst -999999999)))
```

```
(defun delete-max (lst)
  (let ((mx (max-atom lst)))
    (labels ((filter (l)
      (cond
        ((null l) nil)
        ((= (car l) mx)
         (filter (cdr l)))
        (t (cons (car l)
                  (filter (cdr l)))))))
      (filter lst))))
```

```
(print (delete-max '(1 4 2 4 3)))
```

```
->(1 2 3)
```

Max([x]) = x
Max(x :: xs) = max(x, Max(xs))

DelMax([], M) = []
DelMax(x :: xs, M) =
if x = M
then DelMax(xs, M)
else x :: DelMax(xs, M)

DeleteAllMax(L) = DelMax(L, Max(L))