Seminar 5: Recursive Programming in Lisp

1. Define a function which merges, without keeping the doubles, two sorted linear lists.

Ex: (1 3 4 6) (2 4 7 8) => (1 2 3 4 6 7 8)

Merge(L1, …, Ln, R1, …, Rm) = {

[], n = 0 and m = 0,

L, m = 0,

R, n = 0,

L1 U Merge(L2, …, Ln, R2, …, Rm), L1 = R1,

L1 U Merge(L2, …, Ln, R1, …, Rm), L1 < R1,

R1 U Merge(L1, …, Ln, R2, …, Rm), R1 < L1.

}

; merge (L – list of numbers, R – list of numbers)

(defun merge(l r)

(cond

((and (null l) (null r)) nil)

((null l) r)

((null r) l)

((< (car l) (car r))

(cons (car l) (merge (cdr l) r))

)

((= (car l) (car r))

(cons (car l) (merge (cdr l) (cdr r)))

)

(T

(cons (car r) (merge l (cdr r))))

)

)

)

Functions for creating lists: cons, list, append

|  |  |  |  |
| --- | --- | --- | --- |
|  | cons | list | append |
| ‘A ‘B | (A . B) | (A B) | Error |
| ‘A ‘(B C D) | (A B C D) | (A (B C D)) | Error |
| ‘(A B C) ‘(D E F) | ((A B C) D E F) | ((A B C) (D E F)) | (A B C D E F) |
| ‘(A B C) ‘D | ((A B C) . D) | ((A B C) D) | (A B C . D) |
| ‘A ‘B ‘C ‘D | Error | (A B C D) | Error |
| ‘(A B) ‘(C D) ‘(E F) | Error | ((A B) (C D) (E F)) | (A B C D E F) |
| ‘(A B) ‘C ‘(E F) ‘D | Error | ((A B) C (E F) D) | Error |
| ‘(A B) ‘(E F) ‘D | Error | ((A B) (E F) D) | (A B E F . D) |
| ‘(A B) nil ‘(C D) | Error | ((A B) nil ‘(C D)) | (A B C D) |
| Nil ‘(D E F) | (nil D E F) | (nil (D E F)) | (D E F) |

‘(A B C) ‘D => (append ‘(A B C) (list ‘D)) => (A B C D)

‘A ‘(B C D) => (append (list ‘A) ‘(B C D)) => (A B C D)

(A B (C) D E C) remove all occurrences of C => (A B () D E) => (A B nil D E)

1. Remove all occurrences of an element from a list.

RemoveOcc(l1,…, ln, e){

[], n = 0,

RemoveOcc(l2,..,ln,e) l1 = e,

L1 U RemoveOcc(l2,...ln,e) l1 is an atom

RemoveOcc(l1,e) U RemoveOcc(l2,..,ln,e), otherwise

}

(defun remove\_oc(l e)

(cond

((null l) nil)

((equal (car l) e) (remove\_oc (cdr l) e))

((atom (car l)) (cons (car l) (remove\_oc (cdr l) e) ))

(T (cons (remove\_oc (car l) e ) (remove\_oc (cdr l) e )))

)

)

s

1. Build a list with the positions of the minimum number from a linear list.

Ex: (5 A B 3 D 3 7 11 1 A A 1) => (9 12)

Solution ideas:

* + Have several functions: find the minimum, find the list of positions of a given element,combine the previous two.
  + Find the solution by parsing the list only once.

(5 A B 3 D 3 7 11 1 A A 1) 5 (current min) () (lpos of min) 1 (current index) 0 flag

(A B 3 D 3 7 11 1 A A 1) 5 (1) 2 1

(B 3 D 3 7 11 1 A A 1) 5 (1) 3 1

(3 D 3 7 11 1 A A 1) 5 (1) 4 1

(D 3 7 11 1 A A 1) 3 (4) 5 1

(3 7 11 1 A A 1) 3 (4) 6 1

(7 11 1 A A 1) 3 (4 6) 7 1

(11 1 A A 1) 3 (4 6) 8 1

(1 A A 1) 3 (4 6) 9 1

(A A 1) 1 (9) 10 1

(A 1) 1 (9) 11 1

(1) 1 (9) 12 1

() 1 (9 12) 13 1

=> (9 12)

MinPos(l1l2...ln, cmin, idx, a1a2...am, flag) = { a1a2...am, n = 0,

MinPos(l2...ln, cmin, idx+1, a1a2...am, flag), l1 is not a number

MinPos(l2...ln, l1, idx + 1, idx, 1), flag =0,

MinPos(l2...ln, l1, idx + 1, idx,1), l1 < cmin,

MinPos(l2...ln, cmin, idx+1, a1a2...am, 1), l1 > cmin,

MinPos(l2...ln, cmin,idx+1, a1a2...am U idx , 1) l1 =cmin,

}

(defun minPos (l cmin idx a flag)

(cond

((null l) a)

((not (numberp l)) (minPos (cdr l) cmin (+ idx 1) a flag))

((= flag 0) (minPos (cdr l) (car l) (+ idx 1) (list idx) flag))

((< (car l) cmin) (minPos (cdr l) (car l) (+ idx 1) (list idx) 1))

((> (car l) cmin) (minPos (cdr l) cmin (+ idx 1) a 1))

(T (minPos (cdr l) cmin (+ idx 1) (append a (list idx)) 1))

)

)

MinPosMain(l1...ln) = minPos(l1...ln, nil, 1, (), 0)

(defun minPosMain(l)

(minPos l nil 1 nil 0)

)

MinPosMain(l1...ln, index) = nil, n = 0

minPos(l2...ln, l1, index, (index)), if l1 is a number

MinPosMain(l2...ln, index + 1), l1 is not a number