Name: Astăluș Adrian
Group: 921
Subject: 10
Astălus

A. 
```
(defun F(L)
  (cond
    ((NULL L) NIL)
    (T ( lambda(x)
          (cond
            ((> x 2)(CONS x (F(CDR L))))
            (T x)
          ) (F(CAR L))
    ))))
```

- we use a lambda function instead of the recursive call "(F(CAR L))" in order to avoid repetition. This way, the function will be computed only once, its result being used inside the lambda function

3.

Name: Astălus Adrian
Group: 921
Subject: 010
Astalus


```
% function to get combinations
comb([H|_], 1, [H]).
comb([_|T], K, R):-
    comb(T,K,R).
comb([H|T], K, [H|R]):-
    K > 1,
    K1 is K-1,
    comb(T,K1,R).

% condition_comb(l,k) = ~~comb~~( if Condition(comb(l,k)
% condition_comb(L: list; K: integer)
% flow (i,i,i) (i,i,o)
condition_comb(L,K,C):-
    comb(L,K,C),
    condition(C)

% list_len(l) = 0, if list is empty
%               1+ list_len(l2,...,ln) otherwise
% L: list
% flow (i,o)
list_len([], 0).
list_len([_|T], L):-
    list_len(T, L1),
    L is L1+1.

% all_subsets(L, K, Len) = [], if K > Len
%                  U condition_comb(L,K) + all_subsets(L,K+1,Len)
% L: initial list, K: current len. of subsets, Len: max. len. of the list
% flow (i,i,i,o)
all_subsets(_, K, Len, []):-
            K > Len,!.
all_subsets(L, K, Len, [R|R1]):-
        findall(O1, condition_comb(L,K,O1), R),
        K1 is K+1
        all_subsets(L, K1, Len, R1).
```

Name: Astăluş Adrian
Group: 921
Subject: 010
Astalus


```
% odd_count (l) = 0, if list is empty
%                 1 + odd_count (l2,...,ln) if l1 is odd
%                 odd_count (l2,...,ln) if l1 is even
% flow (i,o)
odd_count ([], 0).
odd_count ([H|T], R1):-
        H mod 2 =:= 1,
        R1 is R+1,
        odd_count (T, R).
odd_count ([H|T], R):-
        H mod 2 =:= 0,
        odd_count (T, R).

% subset_sum(l) = 0, if list is empty
%                 l1 + subset_sum (l2,...,ln) otherwise
% flow (i,o)
subset_sum ([], 0).
subset_sum ([H|T], S1):-
        S1 is S+H,
        subset_sum (T, S).

% sort1(L) = [], if L is empty
%            insert (sort1 (l2,...,ln), l1) otherwise
% flow (i,o)
sort1 ([], []):- !.
sort1 ([H|T], O):-
    sort1 (T, O1),
    insert (O1, H, O).

% insert (L, el) = [el], if list is empty
%                  el U l1,...,ln if el < l1
%                  l1 U insert (l2,...,ln) otherwise
% flow (i,i,o)
insert([], E, [E]):- !.
insert([H|T], E, O):-
    E =< H, !,
    O = [E, H|T].
insert ([H|T], E, O):-
    E > H,
    O = [H|O1],
    insert (T, E, O1).
```

```
% mainfunc(l) = all_subsets(l, 2, list_len(l))
% flow (i,o)
mainfunc(L, R) :-
    list_len(L, Len),
    Sort1(L, L1),
    all_subsets(L1, 2, Len, R).
```


Name: Astălus Adrian
Group: 921
Subject: 010

Astalus

c.

Name: Astăluș Adrian
Group: 921
Subject: 010
Astalus
page 5/5

mathematical model

replace_odd (l, niv, el) =

$$\begin{cases} l, \text{ if } l \text{ is an atom and } niv \% 2 == 0 \\ el, \text{ if } l \text{ is an atom and } niv \% 2 == 1 \\ \text{replace\_odd}(l_1, niv+1, el) \cup \ldots \cup \text{replace\_odd}(l_n, niv+1, el), \text{otherwise} \end{cases}$$

l: our initial list; niv: the curent level (depth) of the list; el: the ~~...~~ value we replace with

```
(defun replace_odd (l niv el)
  (cond
    ((AND (atom l) (equals 0 (mod niv 2))) l)
    ((AND (atom l) (equals 1 (mod niv 2))) el)
    (T (mapcar #'(lambda (x) (replace_odd x (+ niv 1) el)) l ))
))

(defun main-replace (l el)          ; this is a wrapper function for our main
  (replace-odd l -1 el)             ; replace function
)                                   ; we take "niv" to be -1 because ~~initially~~
                                    ; the root level is 0, and initially 'l' is
                                    ; a list => replace_odd will go on the last
                                    branch
```

The function 'replace_odd' will check if the current element 'l' is an atom and if the current level is even or odd. If it is odd, ~~#~~ 'l' will be replaced by 'el'. In case 'l' is not an atom, it means it is a list, and we will use 'mapcar' to apply the 'replace_odd' function to all the elements ~~of~~ the list, ~~also~~ increasing the level (depth) by 1.