Rancan Maria 226 Runcan

A.
```
p([],-1):-!.
p([H|T],Rez):-
    p(T,S),
    aux(H,S,Rez).

aux(_,S,Rez):-
    S<1,
    !,
    Rez is S+2.
aux(H,S,Rez):-
    S<0,
    !,
    Rez is S+H.
aux(_,S,Rez):-
    Rez is S.
```

Runcan Maria 226 Runcavo

B. % genereaza_lista(n) = { [], n = 0
[n] (+) genereaza_lista(n-1), altfel

% model de flux (i,o), determinist
% genereaza_lista(N: intreg, R: lista)
% N - numarul până la care se generează lista 1,..., N
% R - lista cu numerele 1,..., N

genereaza_lista(0, []):-!.
genereaza_lista(N, [N|Rez]):-
    N1 is N-1,
    genereaza_lista(N1, Rez).

% comb(l1... lm, k) = 1. [l1], dacă k=1
2. comb(l2... lm, k)
3. l1 (+) comb(l2... lm, k-1), dacă k>1

% model(i,i,o) nedeterminist
% comb(L: lista, K: întreg, Rez: lista)
% L - lista cu elementele ce apar în combinare
% k - nr. de elem. din combinare
% Rez - o combinare de k elem. din L

comb([H|_], 1, [H]).
comb([_|T], K, Rez):- comb(T, K, Rez).
comb([H|T], K, [H|Rez]):-
    K1 is K-1,
    comb(T, K1, Rez).

% combinari(l1... lm, k) = U comb(l1... lm, k)
% model (i, i, o) determinist
% combinari(L: lista, K: întreg, Rez: lista)
% L - lista pe care facem combinările
% K - nr. de elem. dintr-o combinare
% Rez - lista tuturor combinărilor de k elem.

Runcan Maria 226 ~~Runcan~~

```
combinari(L, k, Rez):-
        findall(X, comb(L, k, X), Rez).
```

% verifica($l_1 \dots l_n$) = { false, daca $n \geq 2$ și $(l_1 - l_2)\%2 = 1$
verifica($l_2 \dots l_n$), dacă $n \geq 2$
adevarat, $n = 1$

```
% model(i,o) determinist
% verifica(L: lista)
% L - lista în care verificăm ca diferența dintre două elem. consecutive
% să fie nr. par
verifica([H]):-!.
verifica([H1|[H2|T]]):-
        R is H1-H2,
        R mod 2 =:= 0,
        !
        ;
        verifica([H2|T]).
```

% main_aux($l_1 \dots l_n$) = { $l_1 \oplus$ main_aux($l_2 \dots l_n$), dacă $n \geq 1$ și verifica(l_1) = T
main_aux($l_2 \dots l_n$), $n \geq 1$ și verifica(l_1) = F
[], $n = 0$

```
% model(i,o) determinist
% main_aux(L: lista, R: lista)
% L - lista de combinari
% R - lista de comb. ce verifica proprietatea
main_aux([],[]):-!.
main_aux([H|T],[H|Rez]):-
        verifica(H),
        !
        ;
        main_aux(T,Rez).
main_aux([_|T],Rez):- main_aux(T,Rez).
```

Runcan Maria 226 ~~Runcan~~

```
% main(m, k) = main_aux(combinari(genereaza_lista(m), k))
% model(i,i,o) determinist
% main(N: intreg, k:intreg, Rez: lista)

main(N, K, Rez):-
        genereaza_lista(N, L),
        combinari(L, K, C),
        main_aux(C, Rez).
% N - ~~nr~~. combinarile cautate contin nr. de la 1 la N
% K - nr. de elemente dintr-o comb.
% Rez - lista combinarilor de cate K care verifica proprietatea
```

Runcan Maria 226 Runcan

c. (defun inlocuire (x nivel k)
    (cond
        ((and (atom x) (equal nivel k)) 0)
        ((atom x) x)
        (t (MAPCAR #' (lambda (y)
                        (inlocuire y (+ nivel 1) k)
)
                   x
)
)
)

; inlocuire (x, nivel, k) = $\begin{cases} 0, \text{ dacă } x \text{ atom și } nivel = k \\ x, \text{ dacă } x \text{ atom} \\ \bigcup_{i=1}^{m} inlocuire(x_i, nivel+1, k), \text{ dacă } x \text{ este lista} \end{cases}$

$x_1 \dots x_m$

; x - lista în care se face inlocuirea
; nivel - nivelul curent pe care ne aflăm
; k - nivelul pe care se face înlocuirea
~~; ex de apel: (inlocuire '(a (x (2 b)) (c (d))) 0 2)~~

(defun main (x k)
    (inlocuire x 0 k)
)

; main (x, k) = inlocuire (x, 0, k)
; ex de apel: (main '(a (x (2 b)) (c (d))) 2)