Brînză Alina-Elena, 921, Subject 024
Brînză A.

A. L - list of numbers, the predicate has the flow model (i, o):

```
f([],-1).
f([H|T],S):-f(T,S1),S1<1,S is S1-H,!.
f([_|T],S):-f(T,S).
```

In order to avoid the recursive call f(T,S) in both clauses we will create an ~~additional~~ auxiliary predicate f-aux(H,S,S1). Its parameters are the first element of the list, the final result and the result of the recursive call. This predicate has as flow model the model (i,o,i).

```
f-aux(H,S,S1):-
    S1<1,
    S is S1-H,!.
f-aux(_,S1,S1).

f([],-1).
f([H|T],S):- f(T,S1),f-aux(H,S,S1).
```

In the f-aux predicate we move all the conditions and computations done in the second and ~~ft~~ third clause from the initial predicate that are not in common.

B. $insertOnEveryPos(l_1, l_2, \dots, l_n, e) = \begin{cases} e, \text{ if } n=0 \\ e \cup l_1 \cup l_2 \cup \dots \cup l_n \\ l_1 \cup insertOnEveryPos(l_2, \dots, l_n, e), \text{otherwise} \end{cases}$

```
% insertOnEveryPos(LST: list, E: atom, R: list)
% Flow model: (i,i,o),(o,i,i),(i,o,i),(o,o,i)
insertOnEveryPos([],E,[E]):-!.
insertOnEveryPos(LST,E,[E|LST]).
insertOnEveryPos([H|T],E,[H|R]):-
    insertOnEveryPos(T,E,R).
```

$arrangements(l_1, l_2, \dots, l_n, k) = \begin{cases} l_1, \text{ if } k=1 \\ arrangements(l_2, \dots, l_n, k) \\ insertOnEveryPos(arrangements(l_2, \dots, l_n, k-1), l_1), \text{otherwise} \end{cases}$

```
% arrangements(LST: list; K: int, R: list)
Flow model: (i,i,o),(i,i,i)
arrangements([H|_],1,[H]).
arrangements([_|T],K,R):-
    arrangements(T,K,R).
arrangements([H|T],K,RR):-
    K>1,
    K1 is K-1,
    arrangements(T,K1,R),
    insertOnEveryPos(R,H,RR).
```

$sum(l_1, l_2, \dots, l_n) = \begin{cases} 0, \text{ if } n=0 \\ l_1 + sum(l_2, \dots, l_n), \text{otherwise} \end{cases}$

```
% sum(LST: list, S: int)
Flow model: (i,o),(i,i)
sum([],0).
sum([H|T],S):
    sum(T,S1),
    S is S1+H.
```

Brînză Alina-Elena, 921, Subject a24
Brînză A.

Cont. B.

oneSol(L, K, S) = arrangements(L, K) if ~~product(arr~~ sum(arrangements(L,K)) = S
L: list, K: int, S: ~~list~~ int, R: list, flow model: (i, i, i, o), (i, i, i, i), (i, i, o, i)

oneSol(L, K, S, R):-
    arrangements(L, K, R),
    sum(R, S).

allSol(L, K, S, RL):-
    findall(R, oneSol(L, K, S, R), RL).

The last function is a wrapper function. Flow model: (i, i, i, o), (i, i, i, i), (i, i, o, i).

Brînză Alina-Elena, 921, Subject 024
Brînză A.

C. tree(node subtree1 subtree2 ...)

Mathematical model:

$$nodesOnLevel(tree, level, k) = \begin{cases} \cancel{nil, \text{ if tree is an}} \\ tree, \text{ if tree is an atom and } level = k \\ nil, \text{ if tree is an atom and } level \neq k \\ \bigcup_{i=1}^{n} nodesOnLevel(subtree_i, level+1, k), \\ \quad \text{where } n = \text{number of subtrees} \end{cases}$$

```
(defun NodesOnLevel (l level k)
    (cond
        ((and (atom l)(= level k) (list l))
        ((atom l) nil)
        (t (mapcan #'(lambda (a)(NodesOnLevel a (+ level 1) k)) l))
    )
)

~~(defun wrapperNodes (l k)~~
    ~~(NodesOnLevel (l -1 k)~~
~~)~~

(defun wrapNodes (l k)
  (NodesOnLevel l -1 k)
)
```

We start with ~~0~~ -1 because when we first call mapcan we will give in this way the level 0 to the root.