%Dându-se o listă formată din numere întregi, să se genereze în PROLOG lista aranjamentelor cu număr par de elemente,
%având suma număr impar. Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.

```
% lungime1(L:list, R:integer)
% L - input list
% R - length of list
% Model: lungime1(L) =
%   1. 0, if L = []
%   2. 1 + lungime1(tail(L))
% Flow model: (i,o) - deterministic
lungime1([],0).
lungime1([_|T],R) :-
    lungime1(T,R1),
    R is R1 + 1.



% main1(L:list, S:list)
% L - input list
% S - resulting list of arrangements
% Model: main1(L) = U {SS | generare1(L,0,N,0,SS)}
% Flow model: (i,o) - deterministic
main1(L,S):-
    lungime1(L,N),
    findall(SS, generare1(L,0,N,0,SS),S).



% generare1(L:list, K:integer, N:integer, SP:integer, SS:list)
% L - current list
% K - current position
% N - total length
% SP - partial sum
% SS - current arrangement
% Model: generare1(L,K,N,SP) =
%   1. [], if K ≤ N, K mod 2 = 0, SP mod 2 = 1
%   2. [E|SS], where E ∈ L, SS = generare1(L-{E}, K+1, N, SP+E)
% Flow model: (i,i,i,i,o) - nondeterministic
generare1(_,K,N,SP,[]) :-
    K =< N,
    K mod 2 =:= 0,
    SP mod 2 =:= 1.
generare1(L,K,N,SP,[E|SS]):-
    K =< N,
    candidatC1(E,L),
    eliminatC1(E,L,NL),
    SP1 is SP + E,
    K1 is K + 1,
    generare1(NL,K1,N,SP1,SS).


% candidatC1(E:element, L:list)
% E - selected element
```

```
% L - input list
% Model: candidatC1(E,L) =
%   1. l1, if E = l1
%   2. candidatC1(E,l2...ln)
% Flow model: (o,i) - nondeterministic
candidatC1(E,[E|_]).
candidatC1(E,[_|T]) :- candidatC1(E,T).



% eliminatC1(E:element, L:list, R:list)
% E - element to remove
% L - input list
% R - resulting list
% Model: eliminatC1(E,L) =
%   1. [], if L = []
%   2. [l1 (+) eliminatC1(E,l2..ln)], if l1 ≠ E
%   3. (l2,..,ln), if l1 = E
% Flow model: (i,i,o) - deterministic
eliminatC1(_, [], []).
eliminatC1(E, [H|T], [H|R]) :-
    E =\= H,
    eliminatC1(E,T,R).
eliminatC1(E, [E|T],T).



%Să se scrie un program PROLOG care generează lista permutărilor mulţimii 1..N, cu proprietatea că valoarea absolută a
%diferenţei între 2 valori consecutive din permutare este >=2. Se vor scrie modelele matematice și modelele de flux pentru
%predicatele folosite.
%Exemplu- pentru N=4 ⇒ [[3,1,4,2], [2,4,1,3]] (nu neapărat în această ordine)

% genereazaN(A:integer, B:integer, L:list)
% Model: genereazaN(a,b) =
%   1. [], if a > b
%   2. [a] + genereazaN(a+1,b), if a ≤ b
% Flow model: (i,i,o) - deterministic
genereazaN(A,B,[]) :- A > B.
genereazaN(A,B,[A|N]) :- A =< B, A1 is A + 1, genereazaN(A1,B,N).


% main2(N:integer, S:list)
% Model: main2(N) = U {SS | generare2(L,N,-N,SS)}
% Flow model: (i,o) - deterministic
main2(N,S) :-
    genereazaN(1,N,L),
    findall(SS, generare2(L,N,-N,SS), D),
    eliminaDuplicate(D,S).


% cauta(E:element, L:list)
% Model: cauta(E,[l1,...,ln]) =
%   1. false, if L = []
```

```
%   2. true, if E = l1
%   3. cauta(E,[l2,...,ln])
% Flow model: (i,i) - deterministic0
cauta(_, []) :- false.
cauta(E, [H|_]) :-
    E == H,true.
cauta(E, [_|T]) :-
    cauta(E,T).

eliminaDuplicate(L:list, R:list)
% Model: eliminaDuplicate([l1,...,ln]) =
%   1. [], if L = []
%   2. eliminaDuplicate([l2,...,ln]), if cauta(l1,[l2,...,ln])
%   3. [l1] + eliminaDuplicate([l2,...,ln]), otherwise
% Flow model: (i,o) - deterministic
eliminaDuplicate([],[]).
eliminaDuplicate([H|T],S) :-
    cauta(H,T),
    eliminaDuplicate(T,S).
eliminaDuplicate([H|T],[H|S]) :-
    \+ cauta(H,T),
    eliminaDuplicate(T,S).


% generare2(L:list, N:integer, U:integer, R:list)
% Model: generare2(L,N,U) =
%   1. [], if N = 0
%   2. [l1|SS], where l1 ∈ L, |l1-U| ≥ 2, SS = generare2(L-{l1},N-1,l1)
% Flow model: (i,i,i,o) - nondeterministic
generare2(_,0,_,[]).

generare2(L,N,-N,[E|SS]) :-
    N > 0,
    candidat2(E,L),
    eliminare2(E,L,NL),
    N1 is N - 1,
    generare2(NL,N1,E,SS).

generare2(L,N,U,[E|SS]) :-
    N > 0,
    candidat2(E,L),
    eliminare2(E,L,NL),
    abs(E - U) >= 2,
    N1 is N - 1,
    generare2(NL,N1,E,SS).

candidat2(E,[E|_]).
candidat2(E,[_|T]) :- candidat2(E,T).

eliminare2(_,[],[]).
eliminare2(E,[H|T],[H|R]) :- E =\= H, eliminare2(E,T,R).
eliminare2(E,[E|T],T).
```

%Să se scrie un program PROLOG care generează lista aranjamentelor de k elemente dintr-o listă de numere întregi, pentru
%care produsul elementelor e mai mic decât o valoare V dată. Se vor scrie modelele matematice și modelele de flux pentru
%predicatele folosite.
%Exemplu- pentru lista [1, 2, 3], k=2 și V=7 ⇒ [[1,2],[2,1],[1,3],[3,1],[2,3],[3,2]] (nu neapărat în această ordine)

```
% main3(L:list, K:integer, V:integer, SOL:list)
% Model: main3(L,K,V) = U {S | generare3(L,K,V,1,S)}
% Flow model: (i,i,i,o) - deterministic
main3(L,K,V,SOL) :-
    findall(S, generare3(L,K,V,1,S),SOL).


% generare3(L:list, K:integer, V:integer, PP:integer, SOL:list)
% Model: generare3(L,K,V,PP) =
%   1. [], if K = 0, PP < V
%   2. [l1|SOL], where l1 ∈ L, SOL = generare3(L-{l1},K-1,V,PP*l1)
% Flow model: (i,i,i,i,o) - nondeterministic
generare3(_,0,V,PP,[]) :-
    PP < V.
generare3(L,K,V,ProdusPartial,[E|SOL]) :-
    K > 0,
    candidat3(E,L),
    eliminare3(E,L,NL),
    PP1 is ProdusPartial * E,
    K1 is K - 1,
    generare3(NL,K1,V,PP1,SOL).


% candidat3(E:element, L:list)
% Model: candidat3(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidat3(E,[l2,...,ln])
% Flow model: (o,i) - nondeterministic
candidat3(E, [E|_]).
candidat3(E, [_|T]) :- candidat3(E,T).


% eliminare3(E:element, L:list, R:list)
% Model: eliminare3(E,[l1,...,ln]) =
%   1. [], if L = []
%   2. [l1|eliminare3(E,[l2,...,ln])], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminare3(_,[],[]).
eliminare3(E,[H|T],[H|R]) :- E =\= H, eliminare3(E,T,R).
eliminare3(E,[E|T],T).
```

%Să se scrie un program PROLOG care generează lista submulţimilor formate cu elemente unei liste listă de numere întregi,

%având număr suma elementelor număr impar și număr par nenul de elemente pare. Se vor scrie modelele matematice și
%modelele de flux pentru predicatele folosite.
%Exemplu- pentru lista [2,3,4] ⇒ [[2,3,4]]

```
% main4(L:list, SS:list)
% Model: main4(L) = U {S | generare4(L,0,0,S)}
% Flow model: (i,o) - deterministic
main4(L,SS) :-
    findall(S,generare4(L,0,0,S),SS).


% generare4(L:list, S:integer, C:integer, SS:list)
% Model: generare4([l1,...,ln],S,C) =
%   1. [], if L = [], S mod 2 = 1, C mod 2 = 0, C > 0
%   2. [l1|SS], if l1 mod 2 = 0, SS = generare4([l2,...,ln],S+l1,C+1)
%   3. [l1|SS], if l1 mod 2 = 1, SS = generare4([l2,...,ln],S+l1,C)
% Flow model: (i,i,i,o) - nondeterministic
generare4([],S,C,[]) :-
    S mod 2 =:= 1,
    C mod 2 =:= 0,
    C > 0.
generare4([H|T],S,C,[H|SS]) :-
    S1 is S + H,
    H mod 2 =:= 0,
    C1 is C + 1,
    generare4(T,S1,C1,SS).
generare4([H|T],S,C,[H|SS]) :-
    S1 is S + H,
    H mod 2 =:= 1,
    generare4(T,S1,C,SS).
generare4([_|T],S,C,SS) :-
    generare4(T,S,C,SS).
```

%Să se scrie un program PROLOG care generează lista submulţimilor de sumă S dată, cu elementele unei liste, astfel încât
%numărul elementelor pare din submulțime să fie par. Exemplu- pentru lista [1, 2, 3, 4, 5, 6, 10] şi S=10 ⇒ [[1,2,3,4], [4,6]].

```
% main5(L:list, S:integer, SF:list)
% Model: main5(L,S) = U {SS | generare5(L,S,0,0,SS)}
% Flow model: (i,i,o) - deterministic
main5(L,S,SF) :-
    findall(SS,generare5(L,S,0,0,SS),SF).


% generare5(L:list, S:integer, SP:integer, C:integer, SS:list)
% L - input list
% S - target sum
% SP - partial sum
% C - count of even numbers
```

```
% SS - current subset
% Model: generare5([l1,...,ln],S,SP,C) =
%   1. [], if L = [], SP = S, C mod 2 = 0
%   2. [l1|SS], if l1 mod 2 = 0, SS = generare5([l2,...,ln],S,SP+l1,C+1)
%   3. [l1|SS], if l1 mod 2 = 1, SS = generare5([l2,...,ln],S,SP+l1,C)
% Flow model: (i,i,i,i,o) - nondeterministic
generare5([],S,SP,C,[]) :-
    C mod 2 =:= 0,
    SP =:= S.
generare5([H|T],S,SP,C,[H|SS]) :-
    SP1 is SP + H,
    SP1 =< S,
    H mod 2 =:= 0,
    C1 is C + 1,
    generare5(T,S,SP1,C1,SS).

generare5([H|T],S,SP,C,[H|SS]) :-
    SP1 is SP + H,
    SP1 =< S,
    H mod 2 =:= 1,
    generare5(T,S,SP1,C,SS).

generare5([_|T],S,SP,C,SS) :-
    generare5(T,S,SP,C,SS).



%Să se scrie un program PROLOG care generează lista submulţimilor cu suma număr impar, cu valori din intervalul [a, b]. Se
%vor scrie modelele matematice și modelele de flux pentru predicatele folosite.
%Exemplu- pentru a=2 și b=4 ⇒ [[2,3],[3,4],[2,3,4]] (nu neapărat în această ordine)


% Model: genereazaAB(A,B) =
%   1. [], if A > B
%   2. [A] + genereazaAB(A+1,B), if A ≤ B
% Flow model: (i,i,o) - deterministic
genereazaAB(A,B,[]) :- A > B.
genereazaAB(A,B,[A|L]) :- A =< B, A1 is A + 1, genereazaAB(A1,B,L).


% main6(A:integer, B:integer, SS:list)
% Model: main6(A,B) = U {S | genereaza6(L,0,S), L = [A..B]}
% Flow model: (i,i,o) - deterministic
main6(A,B,SS) :-
    genereazaAB(A,B,L),
    findall(S,genereaza6(L,0,S),SS).


% genereaza6(L:list, S:integer, R:list)
% Model: genereaza6([l1,...,ln],S) =
%   1. [], if L = [], S mod 2 = 1
%   2. [l1|R], R = genereaza6([l2,...,ln],S+l1)
% Flow model: (i,i,o) - nondeterministic
genereaza6([],S,[]) :-
    S mod 2 =:= 1.
```

```
genereaza6([H|T],S,[H|R]) :-
    S1 is S + H,
    genereaza6(T,S1,R).
genereaza6([_|T],S,R) :-
    genereaza6(T,S,R).
```

%Să se scrie un program PROLOG care generează lista combinărilor de k elemente cu numere de la 1 la N, având diferența
%între două numere consecutive din combinare număr par. Se vor scrie modelele matematice și modelele de flux pentru
%predicatele folosite.
%Exemplu- pentru N=4, k=2 ⇒ [[1,3],[2,4]] (nu neapărat în această ordine)

```
% genereazaAB7(A:integer, B:integer, L:list)
% Model: genereazaAB7(A,B) =
%   1. [], if A > B
%   2. [A|R], where R = genereazaAB7(A+1,B)
% Flow model: (i,i,o) - deterministic
genereazaAB7(A,B,[]) :- A > B.
genereazaAB7(A,B,[A|R]) :- A =< B, A1 is A + 1, genereazaAB7(A1,B,R).


% main7(N:integer, K:integer, CS:list)
% Model: main7(N,K) = U {S | genereaza7(L,K,N,-N,S), L = [1..N]}
% Flow model: (i,i,o) - deterministic
main7(N,K,CS) :-
    genereazaAB7(1,N,L),
    findall(S,genereaza7(L,K,N,-N,S),D),
    stergeRep(D,CS).


% cauta7(L:list, E:element)
% L - list to search in
% E - element to find
% Model: cauta7([l1,...,ln], E) =
%   1. false, if L = []
%   2. true, if E = l1
%   3. cauta7([l2,...,ln], E)
% Flow model: (i,i) - deterministic
cauta7([],_) :- false.
cauta7([H|_],E) :-
    H == E,
    true.
cauta7([_|T],E) :-
    cauta7(T,E).


% stergeRep(L:list, R:list)
% L - input list with possible duplicates
% R - resulting list without duplicates
% Model: stergeRep([l1,...,ln]) =
%   1. [], if L = []
%   2. stergeRep([l2,...,ln])
%   3. [l1|stergeRep([l2,...,ln])],altfel
```

```
% Flow model: (i,o) - deterministic
stergeRep([],[]).
stergeRep([H|T],R) :-
    cauta7(T,H),
    stergeRep(T,R).
stergeRep([H|T],[H|R]) :-
    \+ cauta7(T,H),
    stergeRep(T,R).



% genereaza7(L:list, K:integer, N:integer, U:integer, R:list)
% Model: genereaza7(L,K,N,U) =
%   1. [], if K = 0
%   2. [E|R], if K > 0, E ∈ L, (U = -N or (E > U and |U-E| mod 2 = 0))
% Flow model: (i,i,i,i,o) - nondeterministic

genereaza7(_,0,_,_,[]).
genereaza7(L,K,N,U,[E|R]):-
    K > 0,
    candidat7(E,L),
    eliminare7(L,E,NL),
    K1 is K - 1,
    U =:= -N,
    genereaza7(NL,K1,N,E,R).

genereaza7(L,K,N,U,[E|R]):-
    K > 0,
    candidat7(E,L),
    eliminare7(L,E,NL),
    K1 is K - 1,
    E > U,
    abs(U - E) mod 2 =:= 0,
    genereaza7(NL,K1,N,E,R).

% candidat7(E:element, L:list)
% E - element to be selected
% L - input list
% Model: candidat7(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidat7(E,[l2,...,ln])
% Flow model: (o,i) - nondeterministic
candidat7(E,[E|_]).
candidat7(E,[_|T]) :- candidat7(E,T).

% eliminare7(L:list, E:element, R:list)
% L - input list
% E - element to remove
% R - resulting list
% Model: eliminare7([l1,...,ln],E) =
%   1. [], if L = []
%   2. [l1|eliminare7([l2,...,ln],E)], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminare7([],_,[]).
eliminare7([H|T],E,[H|R]) :-
    H =\= E,
```

```
    eliminare7(T,E,R).
eliminare7([E|T],E,T).

%Să se scrie un program PROLOG care generează lista submulţimilor formate cu elemente unei liste listă de numere întregi,
%având suma elementelor număr impar și număr impar de elemente impare. Se vor scrie modelele matematice și modelele
%de flux pentru predicatele folosite.


% main8(L:list, SS:list)
% L - input list
% SS - resulting list of subsets
% Model: main8(L) = U {S | generare8(L,0,0,S)}
% Flow model: (i,o) - deterministic
main8(L,SS):-
    findall(S,generare8(L,0,0,S),SS).

% generare8(L:list, S:integer, C:integer, R:list)
% L - current list
% S - partial sum
% C - count of odd numbers
% R - current subset
% Model: generare8([l1,...,ln],S,C) =
%   1. [], if L = [], S mod 2 = 1, C mod 2 = 1
%   2. [l1|R], if l1 mod 2 = 1, R = generare8([l2,...,ln],S+l1,C+1)
%   3. [l1|R], if l1 mod 2 = 0, R = generare8([l2,...,ln],S+l1,C)
% Flow model: (i,i,i,o) - nondeterministic

generare8([],S,C,[]) :-
    S mod 2 =:= 1,
    C mod 2 =:= 1.
generare8([H|T],S,C,[H|R]) :-
    S1 is S + H,
    H mod 2 =:= 1,
    C1 is C + 1,
    generare8(T,S1,C1,R).
generare8([H|T],S,C,[H|R]) :-
    S1 is S + H,
    H mod 2 =:= 0,
    generare8(T,S1,C,R).
generare8([_|T],S,C,R) :-
    generare8(T,S,C,R).

%Dându-se o listă formată din numere întregi, să se genereze în PROLOG lista submulţimilor cu număr par de elemente. Se
%vor scrie modelele matematice și modelele de flux pentru predicatele folosite.
%Exemplu- pentru lista L=[2,3,4] ⇒  [[],[2,3],[2,4],[3,4]] (nu neapărat în această ordine)

% main9(L:list, S:list)
% L - input list
% S - resulting list of subsets
% Model: main9(L) = U {SS | generare9(L,0,SS)}
% Flow model: (i,o) - deterministic

main9(L,S) :-
    findall(SS,generare9(L,0,SS),S).
```

```
% generare9(L:list, C:integer, R:list)
% Model: generare9([l1,...,ln],C) =
%   1. [], if L = [], C mod 2 = 0
%   2. [l1|R], R = generare9([l2,...,ln],C+1)
%   3. generare9([l2,...,ln],C)
% Flow model: (i,i,o) - nondeterministic
generare9([],C,[]) :-
    C mod 2 =:= 0.
generare9([H|T],C,[H|R]) :-
    C1 is C + 1,
    generare9(T,C1,R).
generare9([_|T],C,R) :-
    generare9(T,C,R).

%Dându-se o listă formată din numere întregi, să se genereze în PROLOG lista submulţimilor cu cel puţin N elemente având
%suma divizibilă cu 3. Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.
%Exemplu- pentru lista L=[2,3,4] și N=1 ⇒  [[3],[2,4],[2,3,4]] (nu neapărat în această ordine)

% main10(L:list, N:integer, Sol:list)
% Model: main10(L,N) = U {S | generare10(L,0,N,0,S)}
% Flow model: (i,i,o) - deterministic
main10(L, N, Sol) :-
    findall(S, generare10(L,0,N,0,S), Sol).


% generare10(L:list, C:integer, N:integer, S:integer, R:list)
% Model: generare10([l1,...,ln],C,N,S) =
%   1. [], if L = [], S mod 3 = 0, C ≥ N
%   2. [l1|R], R = generare10([l2,...,ln],C+1,N,S+l1)
%   3. generare10([l2,...,ln],C,N,S)
% Flow model: (i,i,i,i,o) - nondeterministic
generare10([],C,N,S,[]) :-
    S mod 3 =:= 0,
    C >= N.
generare10([H|T],C,N,S,[H|R]) :-
    S1 is S + H,
    C1 is C + 1,
    generare10(T,C1,N,S1,R).
generare10([_|T],C,N,S,R) :-
    generare10(T,C,N,S,R).


%C. Să se scrie un program PROLOG care generează lista combinărilor de  k elemente dintr-o listă de numere întregi, având
%suma număr par. Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.
%Exemplu- pentru lista [6, 5, 3, 4], k=2 ⇒ [[6,4],[5,3]] (nu neapărat în această ordine)

% mainX(L:list, K:integer, S:list)
% L - input list
% K - number of elements
% S - resulting combinations
% Model: mainX(L,K) = U {V | generareX(L,0,K,0,-K,V)}
% Flow model: (i,i,o) - deterministic
mainX(L,K,S):-
    findall(V,generareX(L,0,K,0,-K,V),S).
```

```
% generareX(L:list, C:integer, K:integer, S:integer, U:integer, V:list)
% L - current list
% C - current count
% K - target count
% S - partial sum
% U - last element
% V - current combination
% Model: generareX(L,C,K,S,U) =
%   1. [], if C = K, S mod 2 = 0
%   2. [E|V], if C < K, E ∈ L, (U = -K or E < U), V = generareX(L-{E},C+1,K,S+E,E)
% Flow model: (i,i,i,i,i,o) - nondeterministic
generareX(_,C,K,S,_,[]) :-
    C =:= K, S mod 2 =:= 0.
generareX(L,C,K,S,U,[E|V]) :-
    C < K,
    candidatX(E,L),
    eliminareX(L,E,NL),
    C1 is C + 1,
    S1 is S + E,
    U =:= -K,
    generareX(NL,C1,K,S1,E,V).
generareX(L,C,K,S,U,[E|V]) :-
    C < K,
    candidatX(E,L),
    eliminareX(L,E,NL),
    C1 is C + 1,
    S1 is S + E,
    E < U,
    generareX(NL,C1,K,S1,E,V).


% candidatX(E:element, L:list)
% E - element to be selected
% L - input list
% Model: candidatX(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidatX(E,[l2,...,ln])
% Flow model: (o,i) - nondeterministic
candidatX(E,[E|_]).
candidatX(E,[_|T]) :- candidatX(E,T).


% eliminareX(L:list, E:element, R:list)
% L - input list
% E - element to remove
% R - resulting list
% Model: eliminareX([l1,...,ln],E) =
%   1. [], if L = []
%   2. [l1|eliminareX([l2,...,ln],E)], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminareX([],_,[]).
eliminareX([H|T],E,[H|R]) :- H =\= E, eliminareX(T,E,R).
eliminareX([E|T],E,T).
```

%C. Scrieţi un program PROLOG care determină dintr-o listă formată din numere întregi lista subşirurilor cu cel puțin 2
%elemente, formate din elemente în ordine strict crescătoare. Se vor scrie modelele matematice și modelele de flux pentru
%predicatele folosite.
%Exemplu- pentru lista [1, 8, 6, 4] ⇒ [[1,8],[1,6],[1,4],[6,8],[4,8],[4,6],[1,4,6],[1,4,8],[1,6,8],[4,6,8],[1,4,6,8]] (nu
%neapărat în această ordine)

```
% main11(L:list, R:list)
% L - input list
% R - resulting list of subsequences
% Model: main11(L) = U {S | generare11(L,-inf,[],S), length(S) >= 2}
% Flow model: (i,o) - deterministic
main11(L,R) :-
    findall(S, (generare11(L,-999999,[],S), length(S,Len), Len >= 2), R).

% generare11(L:list, Last:number, Acc:list, R:list)
% L - remaining elements to process
% Last - last element added to sequence
% Acc - accumulator for current sequence
% R - resulting subsequence
% Model: generare11([l1,...,ln], Last, Acc) =
%   1. Acc, if L = []
%   2. generare11([l2,...,ln], Last, Acc)
%   3. generare11([l2,...,ln], l1, [l1|Acc]), if l1 > Last
% Flow model: (i,i,i,o) - nondeterministic
generare11([], _, Acc, R) :-
    reverse(Acc, R).
generare11([H|T], Last, Acc, R) :-
    H > Last,
    generare11(T, H, [H|Acc], R).
generare11([_|T], Last, Acc, R) :-
    generare11(T, Last, Acc, R).
```

%C. Pentru o valoare N dată, să se genereze lista permutărilor cu elementele N, N+1,….,2*N-1 având proprietatea că valoarea
%absolută a diferenţei dintre două valori consecutive din permutare este <=2. Se vor scrie modelele matematice și modelele
%de flux pentru predicatele folosite.

```
generareazaElem(A,B,[]) :- A > B.
generareazaElem(A,B,[A|L]) :- A =< B, A1 is A + 1,generareazaElem(A1,B,L).

% main18(N:integer, S:list)
% N - starting number
% S - resulting permutations
% Model: main18(N) = U {D | generare18(L,N,-N,D), L = [N..2N-1]}
% Flow model: (i,o) - deterministic
main18(N,S) :-
    generareazaElem(N, 2 * N-1,L),
    findall(D,generare18(L,N,-N,D),S).
```

```
% generare18(L:list, N:integer, U:integer, D:list)
% L - current list
% N - remaining elements
% U - last element
% D - current permutation
% Model: generare18(L,N,U) =
%   1. [], if N = 0
%   2. [E|D], if N > 0, E ∈ L, (U = -N or |E-U| ≤ 2), D = generare18(L-{E},N-1,E)
% Flow model: (i,i,i,o) - nondeterministic
generare18([],0,_,[]).
generare18(L,N,U,[E|D]) :-
    N > 0,
    candidat18(E,L),
    eliminare18(L,E,NL),
    N1 is N - 1,
    U =:= -N,
    generare18(NL,N1,E,D).

generare18(L,N,U,[E|D]) :-
    N > 0,
    candidat18(E,L),
    eliminare18(L,E,NL),
    N1 is N - 1,
    abs(E - U) =< 2,
    generare18(NL,N1,E,D).


% candidat18(E:element, L:list)
% E - element to be selected
% L - input list
% Model: candidat18(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidat18(E,[l2,...,ln])
% Flow model: (o,i) - nondeterministic
candidat18(E,[E|_]).
candidat18(E,[_|T]) :- candidat18(E,T).


% eliminare18(L:list, E:element, R:list)
% L - input list
% E - element to remove
% R - resulting list
% Model: eliminare18([l1,...,ln],E) =
%   1. [], if L = []
%   2. [l1|eliminare18([l2,...,ln],E)], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminare18([],_,[]).
eliminare18([H|T],E,[H|R]) :- H =\= E, eliminare18(T,E,R).
eliminare18([E|T],E,T).

%C. Dându-se o listă formată din numere întregi, să se genereze în PROLOG lista permutărilor având proprietatea că valoarea
%absolută a diferenţei dintre două valori consecutive din permutare este <=3. Se vor scrie modelele matematice și modelele
%de flux pentru predicatele folosite.
```

```
%Exemplu- pentru lista L=[2,7,5] ⇒ [[2,5,7], [7,5,2]] (nu neapărat în această ordine)

% lungime19(L:list, R:integer)
% L - input list
% R - length of list
% Model: lungime19([l1,...,ln]) =
%   1. 0, if L = []
%   2. 1 + lungime19([l2,...,ln])
% Flow model: (i,o) - deterministic
lungime19([],0).
lungime19([_|T],R) :- lungime19(T,R1), R is R1 + 1.

% main19(L:list, SS:list)
% L - input list
% SS - resulting permutations
% Model: main19(L) = U {F | generare19(L,N,0,-N,F)}
% Flow model: (i,o) - deterministic
main19(L,SS):-
    lungime19(L,N),
    findall(F, generare19(L,N,0,-N,F),SS).


% generare19(L:list, N:integer, C:integer, U:integer, F:list)
% L - current list
% N - total length
% C - current count
% U - last element
% F - current permutation
% Model: generare19(L,N,C,U) =
%   1. [], if C = N
%   2. [E|D], if C < N, E ∈ L, (U = -N or |E-U| ≤ 3), D = generare19(L-{E},N,C+1,E)
% Flow model: (i,i,i,i,o) - nondeterministic
generare19(_,N,C,_,[]) :-
    C =:= N.
generare19(L,N,C,U,[E|D]) :-
    C < N,
    candidat19(E,L),
    eliminare19(L,E,NL),
    C1 is C + 1,
    U =:= -N,
    generare19(NL,N,C1,E,D).

generare19(L,N,C,U,[E|D]) :-
    C < N,
    candidat19(E,L),
    eliminare19(L,E,NL),
    C1 is C + 1,
    abs(U - E) =< 3,
    generare19(NL,N,C1,E,D).


% candidat19(E:element, L:list)
% E - element to select
% L - input list
% Model: candidat19(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidat19(E,[l2,...,ln])
```

```
% Flow model: (o,i) - nondeterministic
candidat19(E,[E|_]).
candidat19(E,[_|T]) :- candidat19(E,T).


% eliminare19(L:list, E:element, R:list)
% L - input list
% E - element to remove
% R - resulting list
% Model: eliminare19([l1,...,ln],E) =
%   1. [], if L = []
%   2. [l1|eliminare19([l2,...,ln],E)], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminare19([],_,[]).
eliminare19([H|T],E,[H|R]) :- H =\= E, eliminare19(T,E,R).
eliminare19([E|T],E,T).

%C. Să se scrie un program PROLOG care generează lista submulţimilor cu  N elemente, cu elementele unei liste, astfel încât
%suma elementelor dintr-o submulțime să fie număr par. Se vor scrie modelele matematice și modelele de flux pentru
%predicatele folosite.
%Exemplu- pentru lista L=[1, 3, 4, 2] şi N=2 ⇒ [[1,3], [2,4]]


% main20(L:list, N:integer, R:list)
% L - input list
% N - required number of elements
% R - resulting subsets
% Model: main20(L,N) = U {S | generare20(L,N,0,0,S)}
% Flow model: (i,i,o) - deterministic
main20(L, N, R) :-
    findall(S, generare20(L,N,0,0,S), R).

% generare20(L:list, N:integer, S:integer, C:integer, R:list)
% L - current list
% N - required elements
% S - partial sum
% C - current count
% R - current subset
% Model: generare20([l1,...,ln],N,S,C) =
%   1. [], if L = [], C = N, S mod 2 = 0
%   2. [l1|R], R = generare20([l2,...,ln],N,S+l1,C+1)
%   3. generare20([l2,...,ln],N,S,C)
% Flow model: (i,i,i,i,o) - nondeterministic
generare20([], N, S, C, []) :-
    C =:= N,
    S mod 2 =:= 0.
generare20([H|T], N, S, C, [H|R]) :-
    S1 is S + H,
    C1 is C + 1,
    generare20(T,N,S1,C1,R).
generare20([_|T], N, S, C, R) :-
    generare20(T,N,S,C,R).
```

%C. Să se scrie un program PROLOG care generează lista aranjamentelor de k elemente dintr-o listă de numere întregi, având
%produs P dat. Se vor scrie modelele matematice și modelele de flux pentru predicatele folosite.
%Exemplu- pentru lista [2, 5, 3, 4, 10], k=2 şi P=20 ⇒ [[2,10],[10,2],[5,4],[4,5]] (nu neapărat în această ordine)

```
% main21(L:list, K:integer, P:integer, S:list)
% L - input list
% K - number of elements
% P - target product
% S - resulting arrangements
% Model: main21(L,K,P) = U {SS | generare21(L,K,1,P,SS)}
% Flow model: (i,i,i,o) - deterministic
main21(L,K,P,S) :-
    findall(SS, generare21(L,K,1,P,SS),S).

% generare21(L:list, K:integer, PP:integer, P:integer, SS:list)
% L - current list
% K - remaining elements
% PP - partial product
% P - target product
% SS - current arrangement
% Model: generare21(L,K,PP,P) =
%   1. [], if K = 0, PP = P
%   2. [l1|SS], where l1 ∈ L, SS = generare21(L-{l1},K-1,PP*l1,P)
% Flow model: (i,i,i,i,o) - nondeterministic

generare21(_,0,PP,P,[]):-
    PP =:= P.
generare21(L,K,PP,P,[E|SS]):-
    K > 0,
    candidat21(E,L),
    eliminare21(L,E,NL),
    P1 is PP * E,
    K1 is K - 1,
    generare21(NL,K1,P1,P,SS).



% candidat21(E:element, L:list)
% Model: candidat21(E,[l1,...,ln]) =
%   1. l1, if E = l1
%   2. candidat21(E,[l2,...,ln])
% Flow model: (o,i) - nondeterministic
candidat21(E,[E|_]).
candidat21(E,[_|T]) :- candidat21(E,T).


% eliminare21(L:list, E:element, R:list)
% Model: eliminare21([l1,...,ln],E) =
%   1. [], if L = []
%   2. [l1|eliminare21([l2,...,ln],E)], if l1 ≠ E
%   3. [l2,...,ln], if l1 = E
% Flow model: (i,i,o) - deterministic
eliminare21([],_,[]).
eliminare21([H|T],E,[H|R]) :- H =\= E, eliminare21(T,E,R).
```

```
eliminare21([E|T],E,T).
```