

P3 - FILIMON BRIANA, 923

9. Generate all permutation of N (N - given) respecting the property: for every  $2 \leq i \leq n$  exists an  $1 \leq j \leq i$ , so  $|v(i)-v(j)|=1$

```
% Student exercise profile
:- set_prolog_flag(occurs_check, error).      % disallow cyclic terms
:- set_prolog_stack(global, limit(8 000 000)). % limit term space (8Mb)
:- set_prolog_stack(local, limit(2 000 000)). % limit environment space

% Your program goes here
%
valid_permutation(N, Perm) :-
    numlist(1, N, Numbers),
    permute(Numbers, [], RevPerm),
    reverse(RevPerm, Perm).

permute([], Perm, Perm).
permute(Available, Current, Perm) :-
    select(X, Available, Rest),
    ( Current = [] -> true
    ; has_neighbor(X, Current)
    ),
    permute(Rest, [X|Current], Perm).

has_neighbor(X, List) :-
    member(Y, List),
    Diff is abs(X - Y),
    Diff =:= 1.

/** <examples> Your example queries go here, e.g.
?- member(X, [cat, mouse]).
```

The screenshot shows a Prolog interface with three distinct query sections:

- Query 1:** `?- valid_permutation(3,P).` This section displays four permutations of length 3: `P = [1, 2, 3]`, `P = [2, 1, 3]`, `P = [2, 3, 1]`, and `P = [3, 2, 1]`. Below the results are buttons for "Next", "10", "100", "1,000", and "Stop".
- Query 2:** `?- valid_permutation(4,P).` This section displays eight permutations of length 4: `P = [1, 2, 3, 4]`, `P = [2, 1, 3, 4]`, `P = [2, 3, 1, 4]`, `P = [2, 3, 4, 1]`, `P = [3, 2, 1, 4]`, `P = [3, 2, 4, 1]`, `P = [3, 4, 2, 1]`, and `P = [4, 3, 2, 1]`.
- Query 3:** `?- valid_permutation(4,P).` This section is currently empty, indicating no results have been displayed yet.

### Mathematical Model

```
% valid_permutation(N, Perm) - generates valid permutations of [1..N]
% valid_permutation(N, Perm) =
% Let Numbers = [1, 2, ..., N]
% permute(Numbers, [], RevPerm) ∧ reverse(RevPerm, Perm)

% permute(Available, Current, Perm) - builds permutation with neighbor constraint
% permute([], Current, Perm) = Perm = Current (base case)
% permute([a1, ..., ak], Current, Perm) =
% ∃ ai ∈ [a1, ..., ak] such that:
% (Current = [] ∨ has_neighbor(ai, Current)) ∧
```

```
%    permute([a1, ..., ai-1, ai+1, ..., ak], [ai | Current], Perm)

% has_neighbor(X, List) - checks if X has an adjacent value in List
% has_neighbor(X, [l1, ..., lm]) =
%   ∃ lj ∈ [l1, ..., lm] : |X - lj| = 1
```