AIT Semantic and Declarative Technologies Course
Homework P2: Prolog data structures and arithmetic

In the problem set below we use the data structure *itree* (integer tree) which is defined as follows.
A Prolog term is an *itree* if and only if:

- it is a leaf, i.e., a `leaf(`n`)` compound, where n is an integer; or
- it is a node, i.e., a `node(`t_1`,`t_2`)` compound, where the two arguments – t_1 and t_2 – are both *itrees*.

We will refer to this data structure simply as a *tree* or as a *binary tree*.
For each task write a Prolog predicate that corresponds to the specification provided as a comment. You can use built-in predicates mentioned in the lecture slides so far, e.g. `number(Term)` and `Y is Expr`.

A sample task with solution

Given a binary tree, count the number of nodes in it, i.e., calculate the number of occurrences of the compound `node(`t_1`,`t_2`)` within the tree data structure.

```
  % tree_node_count(+Tree, ?N): Binary tree Tree has N nodes.

  | ?- tree_node_count(leaf(1), N).
  N = 0 ? ; no
  | ?- tree_node_count(leaf(1), 0).
  yes
  | ?- tree_node_count(leaf(1), 1).
  no
  | ?- tree_node_count(node(leaf(1),node(leaf(2),leaf(3))), N).
  N = 2 ? ; no
  | ?- tree_node_count(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
  N = 3 ? ; no
```

A solution:

```
% tree_node_count(+Tree, ?N): Binary tree Tree has N nodes.
tree_node_count(leaf(_), 0).
tree_node_count(node(L, R), N):-
        tree_node_count(L, N1),
        tree_node_count(R, N2),
        N is N1 + N2 + 1.
```

1. Incrementing all leaf values of a tree

```
    % increment_tree(+Tree0, ?Tree): Tree is obtained from binary tree
    % Tree0 by incrementing each leaf value by 1.

    | ?- increment_tree(leaf(1), Tree).
    Tree = leaf(2) ? ; no
    | ?- increment_tree(node(leaf(1),node(leaf(2),leaf(3))), Tree).
    Tree = node(leaf(2),node(leaf(3),leaf(4))) ? ; no
```

2. Finding the rightmost leaf value

```
    % tree_rightmost_value(+Tree, ?Value): Tree is a binary tree and
    % the integer in its rightmost leaf is Value.

    | ?- tree_rightmost_value(leaf(1), V).
    V = 1 ? ; no
    | ?- tree_rightmost_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), V).
    V = 3 ? ; no
```

3. Finding a leaf value in a tree

   ```
   % tree_leaf_value(+Tree, +V): V is present as a leaf value in Tree.

   | ?- tree_leaf_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), 3).
   yes
   | ?- tree_leaf_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), 5).
   no
   ```

   Most of the time the program for the above task works also for the input-output mode `tree_leaf_value(+Tree, -V)`, i.e., if the second argument is not an integer, but a variable. In such a case the predicate should enumerate all the leaf values, as shown in the example run below. (The order of the solutions does not matter.) Make your code work for this I/O mode as well.

   ```
   | ?- tree_leaf_value(node(node(leaf(1),leaf(4)), node(leaf(2),leaf(3))), V).
   V = 1 ? ; V = 4 ? ; V = 2 ? ; V = 3 ? ; no
   ```

4. Incrementing all elements in a number list

   ```
   % increment_list(+L0, ?L): L is a list of numbers obtained
   % from L0 by incrementing each element by 1.
   % You can assume that L0 is given and is a list of numbers.

   | ?- increment_list([], L).
   L = [] ? ; no
   | ?- increment_list([1.2,5,2], L).
   L = [2.2,6,3] ? ; no
   ```

5. Finding the last element of a list

   There is a library predicate for finding the last element of a list, but please do not use that.

   ```
   % list_last(+List, ?V): List is a list whose last element is V.

   | ?- list_last([], V).
   no
   | ?- list_last([5,1,2,8,7], V).
   V = 7 ? ; no
   ```

6. Finding a value in a list

   There is a built-in predicate for the task below, but please do not use it.

   ```
   % list_element(+List, +V): V is present as an element in the list List.

   | ?- list_element([], x).
   no
   | ?- list_element([a,b,c], a).
   yes
   | ?- list_element([a,b,c,d], c).
   yes
   | ?- list_element([d,c,1,2], 12).
   no
   ```

   Ensure that your code also works for the input-output mode `list_element(+List, ?V)`, by enumerating in `V` all list elements if `V` is a variable. (The order of the solutions does not matter.)

7. Concatenating lists

   The concatenation of lists [x_1, ..., x_k] and [y_1, ..., x_l] is the list [x_1, ..., x_k, y_1, ..., x_l].

   ```
   % list_concat(+L1, +L2, -L3): L3 is the concatenation of lists L1 and L2.

   | ?- list_concat([], [a,1,x], L).
   L = [a,1,x] ? ;
   no
   | ?- list_concat([p,q], [a,1,x], L).
   L = [p,q,a,1,x] ? ;
   no
   ```

   There is a built-in predicate for the above task, but please do not use it.