AIT Semantic and Declarative Technologies Course

Homework P3: Prolog lists, if-then-else

For each problem, write a Prolog predicate that corresponds to the provided head comment. You are free to make use of the predicates defined in the slides and in earlier exercise solutions. Do not use library predicates.

You can define helper predicates. Try to provide the most accurate head comment (specification, in other words) for the helper predicates. Remember that a head comment is an English language sentence which describes the logical relationship between the arguments of the predicate.

Obviously, you are not allowed to use the non-recursive sample implementations shown below. Instead, you should write your own, recursive predicate which is equivalent to the sample implementation.

The following input-output mode indicators will be used in the specs to annotate the arguments:

- `+` – input argument, i.e., it cannot be an unbound variable;
- `-` – output argument, i.e., it has to be an unbound variable;
- `?` – the argument might be either input or output.

When an argument `L` is a list, the mode indicator `+` is normally used to imply that `L` is a proper (closed, not open-ended) list, i.e. not only `L` is not a variable, but no variable can appear in the tail position within the list `L`.

1. Inserting into an ordered list

```
% insert_ord(+RL0, +Element, ?RL):
% RL0 is a proper list of numbers, which is strictly increasing. Element is a given number.
% The task is to return the strictly increasing list RL which is obtained from RL0
% by inserting Element to it, if Element is not already in RL0. Otherwise RL = RL0.

| ?- insert_ord([1,3,5,8], 6, L).          L = [1,3,5,6,8] ? ; no
| ?- insert_ord([1,3,5,8], 3, L).          L = [1,3,5,8] ? ; no
```

   Hint: use the if-then-else construct.

2. Building a list of a given length
   The length of a list is the number of its elements. (Remember, the atom `[]`, when appears as a tail in a list, is not considered an element of the list, it simply signifies that there are no more elements in the list.)

```
% length_list(+Len, ?List): List is a list whose length is Len.

| ?- length_list(3, [1,3,5]).             yes
| ?- length_list(0, L).                   L = [] ? ; no
| ?- length_list(2, L).                   L = [_A,_B] ? ; no
```

   Predicate `length_list/2` could be defined using the BIP `length/2` as:

```
length_list(Len, List) :- length(List, Len).
```

   However, the above code cannot be accepted as a solution, write your own, recursive implementation.

3. Element of a list at a given index

```
% nth1(+N, ?L, ?E): The N-th element of the (possibly open ended) list L
% is E. The head of a list is considered its 1st element. Argument N is
% guaranteed to be a positive integer.

| ?- nth1(3, [a,b,c], E).                  E = c ? ; no
| ?- nth1(3, L, E).                        L = [_A,_B,E|_C] ? ; no
```

   Most probably, your code will work for the case of `L` being an open ended list, even if you don't worry about this. Predicate `nth1/3` could be defined using the BIPs `append/3` and `length/2` as:

```
nth1(N, List, E) :-
    N1 is N-1, length(Front, N1), append(Front, [E|_], List).
```

   However, the above code cannot be accepted as a solution, write your own implementation, using a single recursive predicate.

4. Splitting a list

```
% split(+N, +List, -Front, -Back): Front is the list containing the first
% N elements of List, and Back is the list of the remaining elements of
% List. N can be assumed to be a given integer.

| ?- split(0, [], F, B).                  F = [], B = [] ? ; no
| ?- split(0, [a], F, B).                 F = [], B = [a] ? ; no
| ?- split(1, [a,b,c,d], F, B).           F = [a], B = [b,c,d] ? ; no
| ?- split(3, [a,b,c,d], F, B).           F = [a,b,c], B = [d] ? ; no
| ?- split(4, [a,b,c,d], F, B).           F = [a,b,c,d], B = [] ? ; no
| ?- split(5, [a,b,c,d], F, B).           no
| ?- split(-1, [a,b,c,d], F, B).          no
```

Check what will your predicate do when invoked in mode `split(+N, ?L, ?F, ?B)`. Hopefully, it produces answers like these below. If not, make your code produce such answers.

```
| ?- split(3, List, [a,1,b], Back).       List = [a,1,b|Back] ? ; no
| ?- split(3, List, [a,1|T], Back).       List = [a,1,_A|Back], T = [_A] ? ; no
| ?- split(3, List, Front, Back).         List = [_A,_B,_C|Back], Front = [_A,_B,_C] ? ; no
```

Predicate `split/4` could be defined using the BIPs `append/3` and `length/2` as:

```
split(N, List, Front, Back) :- N >= 0, length(Front, N), append(Front, Back, List).
```

However, this code cannot be accepted as a solution, write your own implementation, as a recursive predicate.

5. Generate all pairings of a list
Given a list `[a1, a2, ..., an]`, we call the term `ai-aj` a *pairing* if $1 \leq i < j \leq n$. (In Prolog, it is quite common to use the operator "`-`" to build pairs.)
The task is to write a predicate which, given a nonempty list, returns the list of all pairings, in arbitrary order. The predicate should fail for an empty list. You can assume that all elements of the input list are distinct.

```
% pairings(+L, -Pairs): Pairs is the list of all pairings in L (in arbitrary order).

| ?- pairings([], Ps).                    no
| ?- pairings([a], Ps).                   Ps = [] ? ; no
| ?- pairings([a,b], Ps).                 Ps = [a-b] ? ; no
| ?- pairings([a,b,c,d], Ps).             Ps = [a-b,a-c,a-d,b-c,b-d,c-d] ? ; no
| ?- pairings([a,b,1,f(a)], Ps).          Ps = [a-b,a-1,a-f(a),b-1,b-f(a),1-f(a)] ? ; no
```

Important: You should **not** use any of the "all-solutions" predicates (`findall`, `bagof`, `setof`).

6. Count the "visible" elements of a list
An element Z of a list of integers L is called left-visible, if for all elements X of L, that occur before Z, $X < Z$ holds.

```
% visible_count(+L, ?N): N is the number of left-visible elements in
% the proper list L of positive integers

| ?- visible_count([1,3,2,1,4,4,5], C).          C = 4 ? ; no
| ?- visible_count([5,1,3,2,1,4,4], C).          C = 1 ? ; no
```

Hint: you may make use of the `max` function supported by all arithmetic BIPs, example:

```
| ?- X is 2*max(5,7), max(X,10)+85 < 100.        X = 14 ? ; no
```