Semantic and Declarative Technologies. Sample mid-course test 1. **Solutions**
Time: 50 minutes. Total score: 200 points

1. Determine whether the execution of the following Prolog queries results in an *error* (no need to name a specific error type), *failure*, or *success*. In case of success, specify the variable substitutions of named (non-void, i.e. non _) variables. All queries are fed to Prolog independently, i.e. typed on their own after the `| ?-` prompt. (6*5 = 30 p)

   (a) `append([], [a|L], [A]).` — `L = [], A = a`

   (b) `[a,b] = [X|Y].` — `X = a, Y = [b]`

   (c) `U+V = 5+7+2.` — `U = 5+7, V = 2`

   (d) `2*3 is X*Y.` — `error`

   (e) `A is 2*4, B = A+1.` — `A = 8, B = 8+1`

   (f) `Z = 1+5, \+ Z = 2*3.` — `Z = 1+5`

2. Assume that the following program is loaded into the Prolog system.

```
p([A,B|_], D, R) :- A < B-D, R = B.
p([A|As], D, R):-   p(As, D, R).
```

   Determine the values that `X` will take as a result of the following (independent) queries. Write down *all* solutions separated by semicolons, in the order Prolog enumerates them. If there are no solutions, write `no`. (4*10+10 = 50 p)

   (a) `p([], 5, X).` — `no`

   (b) `p([1,2], 0, X).` — `2`

   (c) `p([10,12,13,15], 1, X).` — `12 ; 15`

   (d) `p([1,10,9,1,0,42], 10, X).` — `42`

   Consider the following predicate, which builds on the predicate `p/3` defined above:

```
% p(L, Z): Z is an element of L such that...
p(L, Z) :- p(L, 0, Z).
```

   (e) Provide a declarative spec for the predicate `p/2`, i.e. expand the head comment above to a full sentence. Furthermore, describe in what order are the solutions enumerated.

```
% p(L, Z): Z is an element of L such that it is strictly larger than the
%           immediately preceeding element.
%           Solutions are enumerated from left to right.
```

3. Consider a list `L` consisting of integers. We call a non-empty (continuous) sublist `S` of `L` an *all positive segment*, or *aps* for short, if all elements of `S` are positive, and `S` is maximal, i.e. it cannot be extended to a longer sublist containing positive integers only. Write a Prolog procedure which takes a list of integers and returns the first all positive segment. (60 p)

```
% first_aps1(+L, ?S): S is the first all positive segment of integer list L.
first_aps1([X|L], S) :-
        (   X > 0 -> S = [X|S1], pos_prefix1(L, S1)
        ;   first_aps1(L, S)
        ).

% pos_prefix1(L, S): S is the maximal prefix of L consisting of positive integers
pos_prefix1([], []).
pos_prefix1([X|L], S) :-
        (   X > 0 -> S = [X|S1], pos_prefix1(L, S1)
        ;   S = []
        ).
```

```
% first_aps2(+L, ?S): S is the first all positive segment of integer list L.
first_aps2(L, S) :-
        L = [X|_], X > 0, !, pos_prefix2(L, S).
first_aps2([_|L], S) :-
        first_aps2(L, S).

% pos_prefix2(L, S): S is the maximal prefix of L consisting of positive integers
pos_prefix2([X|L], S) :-
        X > 0, !, S = [X|S1], pos_prefix2(L, S1).
pos_prefix2(_, []).

first_aps3([X|L], S) :-
        (   X =< 0 -> first_aps3(L, S)
        ;   S = [X|S1],
            (   L = [Y|_], Y > 0 -> first_aps3(L, S1)
            ;   S1 = []
            )
        ).

first_aps4([X|L], S) :-
        (   X =< 0 -> first_aps4(L, S)
        ;   S = [X|S1],
            (   L == [] -> S1 = []
            ;   L = [Y|_], Y =< 0 -> S1 = []
            ;   first_aps4(L, S1)
            )
        ).

first_aps5([X|L], S) :-
        X =< 0, !, first_aps5(L, S).
first_aps5([X], S) :-
        X > 0, !, S = [X].
first_aps5([X,Y|_], S) :-
        X > 0, Y =< 0, !, S = [X].
first_aps5([X|L], S) :-
        X > 0, L = [Y|_], Y > 0,
        S = [X|S1],
        first_aps5(L, S1).
```

4. Write a Prolog predicate which enumerates all *all-positive-segments* on backtracking. You may use the predicate `first_aps/3`, but you don't have to. (50 p)

```
% aps(+L, ?S): S is an all-positive-segment of integer list L
| ?- aps([1,2,0,3,4], S).                ----> S = [1,2] ? ; S = [3,4] ? ; no
| ?- aps([-1,1,2,0,3,0,4,-3], S).        ----> S = [1,2] ? ; S = [3] ? ; S = [4] ? ; no
| ?- aps([0,-2,-1], S).                  ----> no
| ?- S = [1|_], aps([1,0,2,-1,1,3], S).  ----> S = [1] ? ; S = [1,3] ? ; no

% aps(+L, ?S): S is an all positive segment of integer list L.
aps(L, S) :-
        first_aps(L, S0, R),
        (   S = S0
        ;   aps(R, S)
        ).
```