Semantic and Declarative Technologies. Sample mid-course test.
Time: 80 minutes. Total score: 200 points

For tasks 1 and 2 this is a "closed book" test. You are not allowed to use a computer during this phase. However, you may ask the instructors for clarification or hints, if needed. After submitting solutions for tasks 1 and 2 you can use your laptop for writing the programs for tasks 3 and 4. However you should use internet only for reading the Prolog manuals, slides, and sample programs. Please do not do searches on the internet.

For tasks 1 and 2 you should preferably write the solutions on this task sheet, in the space provided. Otherwise please refer to the (sub)questions using their numeric (and alphabetic) label, e.g., 2.b.

In tasks 3 and 4 you may use all built-in predicates as well as those found in the slides.

When writing down solutions, please refer to the (sub)questions using their numeric (and alphabetic) label, e.g., 2.b.

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]).`

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

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

   (d) `2*3 is X*Y.`

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

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

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).`

   (b) `p([1,2], 0, X).`

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

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

   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.

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 predicate which takes a list of integers and returns the first all positive segment and the list of remaining elements. (70 p)

```
% first_aps(+L, ?S, ?R): S is the first all positive segment of integer
% list L and R is the list of the remaining elements of L following S.

| ?- first_aps([1], S, R).                ----> S = [1], R = [] ? ; no
| ?- first_aps([2,0,3,4], S, R).          ----> S = [2], R = [0,3,4] ? ; no
| ?- first_aps([0,0,-1], S, R).           ----> no
| ?- first_aps([4,5,5,0,1,-1], S, R).     ----> S = [4,5,5], R = [0,1,-1] ? ; no
| ?- first_aps([-1,0,1,-2], S, R).        ----> S = [1], R = [-2] ? ; no
| ?- first_aps([0,1,2,-2,3,4], S, R).     ----> S = [1,2], R = [-2,3,4] ? ; no
| ?- first_aps([0,1,2,-2,3,4], [3,4], R). ----> no
```

   You may define a helper predicate, provided you write a spec (head comment) for it.

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
```