Declarative programming, midterm re-retake, 3rd June, 2005, Budapest
Working time: 90 minutes, total score: 60
Prolog, group „A" (30 points)

When writing a Prolog program, you may use all of the Prolog predicates which were either defined during the lectures or are built-ins. Please refer to the sub-excercises using their corresponding identifier (for example 2.b).

1. What will be the results of the following Prolog goals (error, failure, success)? In case of success, specify the values of the named variables. The goals are given to the Prolog interpreter separately and on their own. (5 points)

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

   (b) `X is 4-3, Y is X+1, Y = 3-1.`

   (c) `D = 3+E, \+ D = 2, R is D+1.`

   (d) `append([],[a,12|_],[A,_]).`

   (e) `2+4-2 = A-B.`

2. Specify the canonical forms (or draw the corresponding tree structures) of the left and right hand sides of the following equations. In the case of named variables, give the resulting variable substitutions. (9 points)

   (a) `[X,[3*_]|Z] = .(Y,[[Y*2]]).`

   (b) `f(_+A*a,[C,_|B],F) = f(F+C,[3*_,b],6).`

3. Let us assume that we have loaded the following program in the Prolog system.

```
p([A,B|_], T, E) :-
   A > T,
   A < B,
   E = A.
p([A|As], _, E):-
   p(As, A, E).
```

   What will the Prolog system answer if we ask the following questions (what will be the substitutions for variable `E`)? Enumerate all solutions in the same order as the system would, separated by a semicolon. If there is no solution, write `{no}`.

   (a) `p([2,4,1.2,3], 0, E).`

   (b) `p([1,5,10], 2, E).`

   (c) `p([3,4,1,5,8], 1, E).`

   (d) `p([3,4,2,5,3,2], 4, E).`

   (e) `p([2,4,6,7,8,2,4,5],3, E).`

   Consider the following predicate which is based on the one above:

```
   % p(L, Z): Z is an element of list L, such that...
   p(L, Z) :- L = [A|As], p(As, A, Z).
```

   (f) Write down the declarative meaning of the predicate `p/2`, i.e., complete the sentence given above. In what order does the predicate give the solutions?

   (8 points)

4. Write a Prolog predicate (`diff`) which gets a list of integers as input and as output it gives back in it's second argument the longest, non extendable prefix of the list, which has all different elements, plus the rest of the list in the third argument. You may use the predicate `all_different/1` to decide if a list contains different elements (you don't have to implement it). You may define auxiliary predicate(s), provided that you give a head comment for them. (8 points)

```
% diff(+L, -D, -R): D is a prefix in L which conatains all
% different elements, R is the rest of the list

| ?-  diff([1,1,1], D, R).          ⇒ D = [1], R = [1,1] ; no
| ?-  diff([1,2,3], D, R).          ⇒ D = [1,2,3], R = [] ; no
| ?-  diff([3,4,2,2], D, R).        ⇒ D = [3,4,2], R = [2] ; no
| ?-  diff([4,5,3,3,2,6], A).       ⇒ D = [4,5,3], R = [3,2,6] ; no
```