Name:

Contact Person at K.U.Leuven (in case of Erasmus students):

Warning : this is just an example of an exam for PLPM. It will be discussed with the students in more detail.

Examination of
Programming languages and Programming Methodologies
date, ??h – ??h

General Guidelines

The examination is **closed book** and takes (at most) 4 hours.
Write your name on every sheet you hand in.
Write on **one** side of the sheet only.
When writing down predicates, add comments that describe their meaning.
When using particular data structures, document them.

Questions

1. Do the following Prolog queries succeed or fail? In case of success, give the bindings of the variables in the query. In case of failure, explain why.

   (a) `?- [ A, B ] = [ [ 2, 3 ] | C]`

   (b) `?- [ 1, A, a ] = [ B, 2 | 5 ]`

   (c) `?- X = 3 + Y, X is 4 .`

   (d) `?- X = [ 3, 4 | Y], Y = Z, Z = [ 2 | T ].`

2. Consider the following Prolog predicate `compress/2`.

```
compress([ ], [ ]) .
compress([ X ], [ X ]) .
compress([ X, X | Xs ], Zs) :- !, compress([ X | Xs ], Zs ).
compress([ X, Y | Xs ], [X | Zs]) :- compress([ Y | Xs ], Zs ).
```

Does the query `?- compress([3,4,4,3], L).` succeed or fail? Sketch its execution by Prolog (e.g. by giving an execution tree). In case of success, indicate how many times it succeeds and give the bindings of the variables (if any) in the query. In case of failure, explain why.

Does the query `?- compress([4,4,4,4], [4,4]) .` succeed or fail? Sketch its execution by Prolog (e.g. by giving an execution tree). In case of success, indicate how many times it succeeds and give the bindings of the variables (if any) in the query. In case of failure, explain why.

Remark: if a query has more than 1 solution, you should give the complete execution trace with all the solutions.

3. **Remark: this exercise is typically similar to one the students have solved during the course or the exercise sessions. It does not have to be about CLP programming.**.
   Write a program that solves the next cryptarithmetic puzzle. Indicate whether or not you are using constraint logic programming over finite domains.

```
      G R E E N
    V I O L E T
+ -----------
    I N D I G O
```

4. Suppose you have to represent a collection of (person, phone number) pairs. Let us assume that such a pair is represented by a Prolog term with functor `tel` and arity 2, e.g. `tel(maria, 2654)`. Note that you can use the builtin predicates `@<`, `@=<`, `@>`, and `@>=` to compare Prolog terms alphabetically, e.g.:

   ```
   ?- jean @< maria.  succeeds.
   ?- tel(jean,2111) @< tel(maria,1111).  succeeds.
   ?- tel(jean,1000) @< tel(jean,1020).  succeeds.
   ?- tel(maria,1000) @< tel(jean,1020).  fails.
   ```

   (a) Give two different data structures that can be used in Prolog to represent sets of the above tel/2 terms. You may assume that every person has only one phone number.

   (b) Discuss their advantages and disadvantages.

(c) Define for both cases the predicate `possadd` that for a given set `S` and for a ground `tel/2` term `X` checks whether `X` is already in the set `S` and if not the predicate should add `X` to `S`.

5. Consider a set of dominoes. Each domino is a rectangle consisting of two parts and each part has a number of spots on it. The number of spots can go from 0 to 6. There are 28 different dominoes. Using four dominoes we can make a configuration as in Figure 1. Note that we used 4 different dominoes and that for each row and column the sum of the spots on the relevant parts of the dominoes is exactly 3, except for the center row and column.

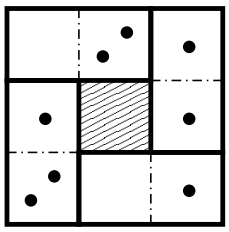

Figure 1: A configuration of dominoes

(a) Define a data structure that represents a domino and a data structure that represents such a configuration of 4 dominoes.

(b) Write a predicate that checks whether a given configuration is a good one.

(c) Write a predicate that succeeds when two given configurations are variants. Note that if we turn the configuration of Figure 1 over for example 180 degrees we get its variant in Figure 2. These kinds of variants should be recognised. Note that you can also turn over 90 and 270 degrees.

(d) Write a predicate that searches all possible good configurations of 4 dominoes, that eliminates the variants and that prints the configurations out in the form

0 2 1 1 1 0 2 1

(i.e. giving the number of spots on the parts of the dominoes starting in the up ward left corner and going clockwise).

Good luck. Gerda Janssens

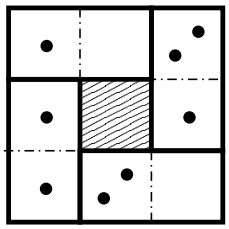

Figure 2: A variant configuration of dominoes