Programming in Prolog - List of Exercises #5

1. Enumerate the main differences between a declarative language such as Prolog and an imperative language such as C.

2. Write a pseudo-code for the execution algorithm used by Prolog.

3. Consider the following program:

```
top(X,Y) :- p(X,Y).                    q(a)
top(X,X) :- s(X).                      q(b).
p(X,Y) :- q(X), r(Y).                  r(c).
p(X,Y) :- s(X), r(Y).                  r(d).
                                       s(e).
```

   Draw the Prolog execution tree for the query `?- top(X,Y)`, showing the respective solutions for each branch of the tree.

4. Consider the following program:

```
p(Y) :- q(X,Y), r(Y).
p(X) :- q(X,X).
q(a,a).
q(a,b).
r(b).
```

   Introduce the cut operator (!) in different places of this code, and comment on the results for query `p(Z)`.

5. What does the program below implement?

```
c(_, [], 0).
c(X, [X|L], N):-
        c(X, L, R), !,
        N is R+1.
c(X, [_|L], N):-
        c(X, L, N).
```

6. Write a Prolog program that can implement the unification algorithm.

7. A 'bug' in the Prolog-to-WAM compiler produced the incomplete and incorrect WAM code as shown in the next page.

   a) Find out what are the missing instructions and the errors.

   b) What does the original Prolog program implement?

   c) Design an instruction or a set of instructions that can reduce the number of instructions needed to execute alternative clauses of this program. Modify the WAM code generated in order to include these new instructions (hint: think about indexing instrutions).

Prolog code:

```
pred(X,[X]).
pred(X,[_|Y]) :- pred(X,Y).
```

WAM code generated by the "buggy" compiler:

```
pred_2:        try_me_else pred_2_2
pred_2_1:      get_variable X1,A1
               proceed
pred_2_2:      get_variable X1,A1
               put_value X1,A1
               put_value X2,A2
               execute pred_2
```

8. What are the basic characteristics that make constraint logic programming languages different from Prolog?

9. Write a small piece of code that fails in Prolog, but can be executed in a constraint logic programming language.

10. Write a Prolog program that can solve the problem:

    ```
    fourty+ten+ten=sixty
    ```

    Implement also a solution using clp(FD).

11. Write a Prolog program that can implement a depth-limited search.

12. Write a concise code in Prolog that can implement a possible movement of the eight-puzzle (in the eight-puzzle, the white cell can move up, right, down, or left, subject to the 3x3 square constraints). Write your code also in a CLP fashion.

13. Suppose you have a database of facts `p(a,4). p(a,1). p(b,2). p(b,3).`. What are the results of executing the following queries for these facts:

    ```
    ?- findall(X,p(Y,X),L).
    ?- bagof(X,p(Y,X),L).
    ?- setof(X,p(Y,X),L).
    ```

    Explain the behavior of each one of these queries.

14. Define a predicate `calc` that prints a prompt, reads an arithmetic expression (without variables), evaluates it, prints the result, and so on until the user enters "quit". You can assume that the user ends each input line with ".". Furthermore, you do not have to handle syntax errors.

15. Why implementing a predicate such as `p(X,Y) :- q(X).`, in a database management context, is not advisable?

16. Using the Prolog built-in predicates that handle red-and-black trees (`https://www.dcc.fc.up.pt/~vsc/Yap/documentation.html#Red_002dBlack-Trees`), write a Prolog program that generates a set of 10000 random numbers, create a red-and-black tree and then randomly deletes all of them. Repeat your experiments for sets of random numbers of sizes 100000 and 500000, and compare execution times. Ideally, plot a graph showing how the execution time varies as we increase the sets of numbers.