Práctica 1

A meta-program is a program that takes another program as data (e.g. compilers). Prolog is particularly suitable for metaprogramming and rapid prototyping (no much emphasis on efficiency). However, once ideas are developed, they can always be re-implementation in a more efficient language.

Ex 1. A simple parser (output: yes/no)

a) What is the language generated by the following grammar?

E → G E1
E1 → + G E1 | ε
G → F G1
G1 → * F G1 | ε
F → (E) | a | b | c | d

b) Implement a top-down parser in Prolog for the grammar. The grammar rules may be replaced by Prolog facts, for instance

```
rule( n(e1), [t(+),n(g),n(e1)] ).
```

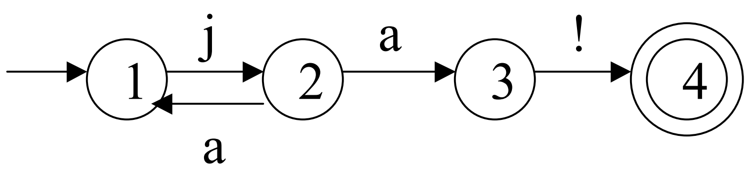
replaces the second rule E1 → + G E1 in the grammar, n(g) represents that symbol g is non-terminal and t(+) represents that symbol + is a terminal symbol.

You are required to implement a predicate parse(Input,Stack) where Input is the input string to be parsed and Stack contains the current string still to be parsed (initially n(e) where e is the main non-terminal). The parser must succeed if the Input string is in the language, and fails otherwise. The basic action of your predicate is to replace a non-terminal on the top of the parse stack (i.e. the front of the list) by the right hand side of the rule defining that non-terminal. If a terminal symbol lies on top of the stack and matches the element t(W) in front of the input, then parsing proceeds by popping t(W) and the symbol on top of the stack (which should also be t(W)), and considering the next element of the input string and next element of the stack. Thus, a string is accepted when the stack and the current string are empty (i.e. both are empty lists). Sample calls to your program are for instance:

```
?- parse([t(+),t(a),t(b)],[n(e)]).                    % +ab  is not in the language
No
?- parse([t(a),t(+),t(b)],[n(e)]).                    % a+b  is in the language
Yes
?- parse([t('('),t(a),t(+),t(b),t(')'),t(*),t(c)],[n(e)]).   % (a+b)*c  is in the language
Yes
```

Ex 2. A simple automata.

Represent the following finite automata in Prolog. Use predicates *initial/1*, *final/1* y *arc/3*. For instance *arc(1,2,j)* represents the first arc. Write predicates for recognizing strings (for instance Prolog must respond 'yes' to *recognize([j,a,j,a,!])*) and generating the accepted strings by the automata (for intance *generate(X)* must return in X one at a time the strings generated by the automata by pressing ';').


$$\begin{tikzcd} \arrow[r] & 1 \arrow[r, bend left, "j"] & 2 \arrow[l, bend left, "a"] \arrow[r, "a"] & 3 \arrow[r, "!"] & \boxed{4} \end{tikzcd}$$

Submitting your answer
The práctica can be solved in teams of two people (1 submission for team). Submission is by email (to rafael.ramirez@upf.edu), the subject of the email must be *'P1- nombre1 apellido1- nombre2 apellido2*. and the message body should contain the answer for 1a) and the programs for 1b) and 2), as well as some tests to your programs. Deadline is the beginning of your next Práctica. Late submissions will be penalized.