```
%------------------------------------------------------------
%Meta data for the file

%file: crypto_i.pro
%line: interpreter for the Crypto language

%some utility programs

%------------------------------------------------------------
:- consult('io.pro').
:- consult('gv.pro').
:- consult('lp.pro').
:- consult('cryptoA.pro').

%DCG for crypto
%------------------------------------------------------------
sentence(problem(numbers(N1,N2,N3,N4,N5), goal(G))) -->
simpleproblemcommand(problem(numbers(N1,N2,N3,N4,N5), goal(G))), [.].
sentence(problem(numbers(N1,N2,N3,N4,N5), goal(G))) -->
simpleproblemquery(problem(numbers(N1,N2,N3,N4,N5), goal(G))), [?].
sentence(problem(numbers(N1,N2,N3,N4,N5), goal(G))) -->
randomproblemcommand(problem(numbers(N1,N2,N3,N4,N5), goal(G))),[.].

simpleproblemcommand(problem(numbers(N1,N2,N3,N4,N5), goal(G))) -->[use],
numberzzz(N1,N2,N3,N4,N5), [to], [make], goal(G),
  {add_crypto_problem_to_KB(N1,N2,N3,N4,N5,G)}.
simpleproblemcommand(problem(numbers(N1,N2,N3,N4,N5), goal(G))) --> [write], goal(G), [in],
[terms], [of], numberzzz(N1,N2,N3,N4,N5),
  {add_crypto_problem_to_KB(N1,N2,N3,N4,N5,G)}.
 % {solve(problem(numbers(N1,N2,N3,N4,N5), goal(G)))}.
randomproblemcommand(problem(numbers(N1,N2,N3,N4,N5), goal(G))) --> [use], [whatever],
[to], [make], [whatever],
  {generate_random_crypto_number(N1),
generate_random_crypto_number(N2),generate_random_crypto_number(N3),generate_rando
m_crypto_number(N4),generate_random_crypto_number(N5),generate_random_crypto_numbe
r(G),add_crypto_problem_to_KB(N1,N2,N3,N4,N5,G)}.

simpleproblemquery(problem(numbers(N1,N2,N3,N4,N5), goal(G)))  --> [can], [you],[make],
goal(G), separator, numberzzz(N1,N2,N3,N4,N5),
  {add_crypto_problem_to_KB(N1,N2,N3,N4,N5,G)}.
  %{solve(problem(numbers(N1,N2,N3,N4,N5), goal(G)))}.
```

```
separator --> [from].
separator --> [with].


goal(N) --> number(N).


numberzzz(N1,N2,N3,N4,N5) --> number(N1), [and], number(N2), [and], number(N3), [and],
number(N4), [and], number(N5).
numberzzz(N1,N2,N3,N4,N5) --> number(N1),number(N2),number(N3), number(N4), [and],
number(N5).
numberzzz(1,2,3,4,5) --> [the], [first], [five], [positive], [numbers].
numberzzz(0,1,2,3,4) --> [numbers], [zero], [through], [four].
numberzzz(1,2,3,4,5) --> [numbers], [one], [through], [five].
numberzzz(2,3,4,5,6) --> [numbers], [two], [through], [six].
numberzzz(3,4,5,6,7) --> [numbers], [three], [through], [seven].
numberzzz(4,5,6,7,8) --> [numbers],[four], [through], [eight].
numberzzz(5,6,7,8,9) --> [numbers],[five], [through], [nine].
numberzzz(1,3,5,7,9) --> [the], [odd], [numbers].
numberzzz(N1,N1,N1,N1,N1) --> [five], pluralnumber(N1).
numberzzz(N1,N1,N1,N1,N2) --> [four], pluralnumber(N1), [and], [one], number(N2).
numberzzz(N1,N2,N2,N2,N2) --> [one], number(N1), [and], [four], pluralnumber(N2).
numberzzz(N1,N1,N2,N2,N2) --> [two], pluralnumber(N1), [and], [three], pluralnumber(N2).
numberzzz(N1,N1,N1,N2,N2) --> [three], pluralnumber(N1), [and], [two], pluralnumber(N2).
numberzzz(N1,N1,N2,N2, N3) --> [two], pluralnumber(N1), [and], [two], pluralnumber(N2), [and],
[one], number(N3).
numberzzz(N1,N2,N2,N3, N3) --> [one], number(N1), [and], [two], pluralnumber(N2), [and],
[two], pluralnumber(N3).
numberzzz(N1,N1,N2,N3,N3) --> [two], pluralnumber(N1), [and],[one], number(N2), [and],[two ],
pluralnumber(N3).

number(0) --> [zero].
number(1) --> [one].
number(2) --> [two].
number(3) --> [three].
number(4) --> [four].
number(5) --> [five].
number(6) --> [six].
number(7) --> [seven].
number(8) --> [eight].
number(9) --> [nine].

pluralnumber(0) --> [zeros].
```

```
pluralnumber(1) --> [ones].
pluralnumber(2) --> [twos].
pluralnumber(3) --> [threes].
pluralnumber(4) --> [fours].
pluralnumber(5) --> [fives].
pluralnumber(6) --> [sixes].
pluralnumber(7) --> [sevens].
pluralnumber(8) --> [eights].
pluralnumber(9) --> [nines].


%The recognizer
%--------------------------------------------------------------------------------
recognizer :-
   read_sentence(S),
   sentence(S, []),
   write('ok'), nl,
   recognizer.
recognizer :-
   write('Not a sentence... '), nl,
   recognizer.


%Testing the augmented DCG -- the parser
% ------------------------------------------------------------------------

parser :-
   read_sentence(S),
   sentence(Problem, S, []),
   write(Problem), nl,
   parser.
parser :-
   write('Not a sentence ...'), nl,
   parser.

%The interpreter
% ------------------------------------------------------------------------

interpreter :-
   read_sentence(S),
   sentence(Problem, S, []),
   solve_without_echo(Problem),
   interpreter.
```

```
interpreter :-
   write('Not a sentence...'), nl,
   interpreter.

% ----------------------------------------------------------------------

%CRYPTO PROBLEM SOLVER -- SOLVE A SPECIFIC PROBLEM BUT WITHOUT THE
ECHO

solve_without_echo(problem(numbers(N1,N2,N3,N4,N5), goal(G))) :-
   add_crypto_problem_to_KB(N1,N2,N3,N4,N5,G),
   solve_problem_decompositionally,
   display_solution.
```