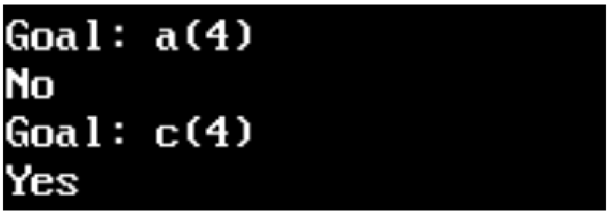
1)Write a program in prolog for cut predicate function

```
predicates

a(integer)

b(integer)

c(integer)

d(integer)

clauses

a(X):-b(X),!,c(X).

a(X):-d(X).

b(1).

c(1).

c(4).

d(3).
```

Output

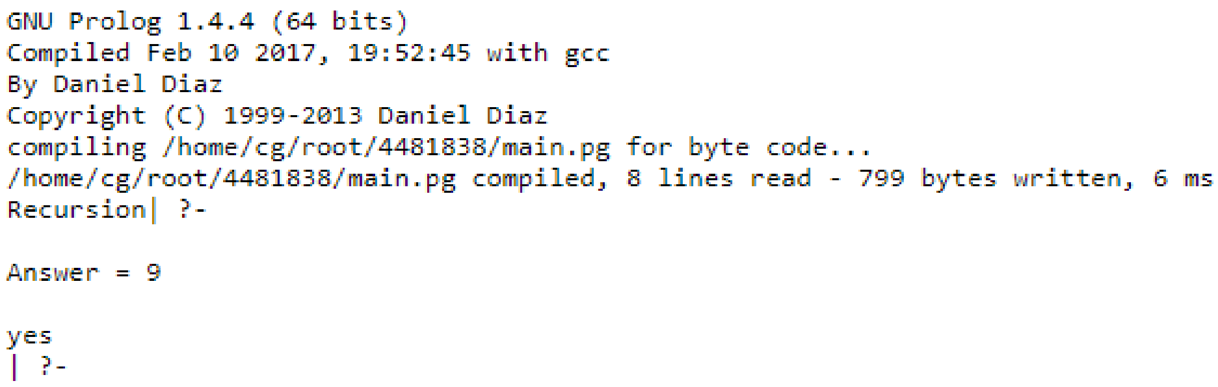


```
Goal: a(4)
No
Goal: c(4)
Yes
```

2)Write a program in prolog for simple recursion.

```
-initialization(main).
main :- write('Recursion').
sumlist([], 0).
sumlist([First | Item], Sum) :-
sumlist(Item, SumOfItem),
Sum is First + SumOfItem.
```

Output.

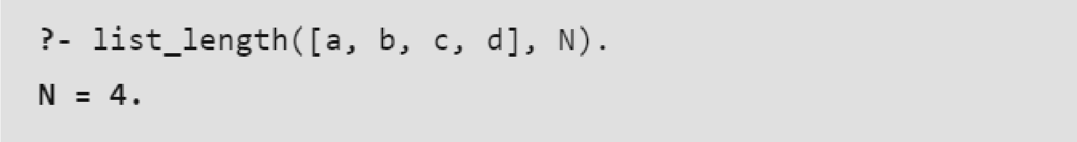


```
GNU Prolog 1.4.4 (64 bits)
Compiled Feb 10 2017, 19:52:45 with gcc
By Daniel Diaz
Copyright (C) 1999-2013 Daniel Diaz
compiling /home/cg/root/4481838/main.pg for byte code...
/home/cg/root/4481838/main.pg compiled, 8 lines read - 799 bytes written, 6 ms
Recursion| ?-

Answer = 9

yes
| ?-
```

3)Write a program in prolog for list predicate function

```
% length of empty list is 0 (base case)

list_length([], 0).

list_length([_ | L], N) :-

   list_length(L, N1),

   N is N1 + 1.

   % If length of L is N1, then length of [_ | L] will be N1 + 1
```

Output

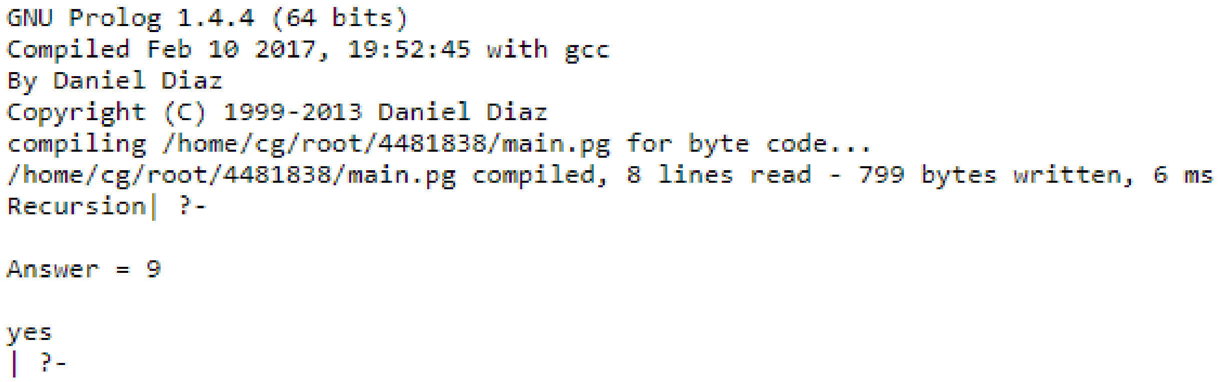


```
?- list_length([a, b, c, d], N).
N = 4.
```

4)Write a program in prolog for simple recursion.

```
initialization(main).
main :- write('Recursion').
sumlist([], 0).
sumlist([First | Item], Sum) :-
sumlist(Item, SumOfItem),
Sum is First + SumOfItem.
```

Output.



```
GNU Prolog 1.4.4 (64 bits)
Compiled Feb 10 2017, 19:52:45 with gcc
By Daniel Diaz
Copyright (C) 1999-2013 Daniel Diaz
compiling /home/cg/root/4481838/main.pg for byte code...
/home/cg/root/4481838/main.pg compiled, 8 lines read - 799 bytes written, 6 ms
Recursion| ?-

Answer = 9

yes
| ?-
```

5)Write a program in prolog for appending a list

```
?- append([a,b], [c], X).
```

output

```
X = [a,b,c].
```

```
?- append(X, [Last], [a,b,c]).
```

output

```
X = [a,b],

Last = c.
```

```
?- append([a,b], More, List).
```

output

```
List = [a,b|More].
```

6)Write a program in prolog to implement medical diagnostic elements

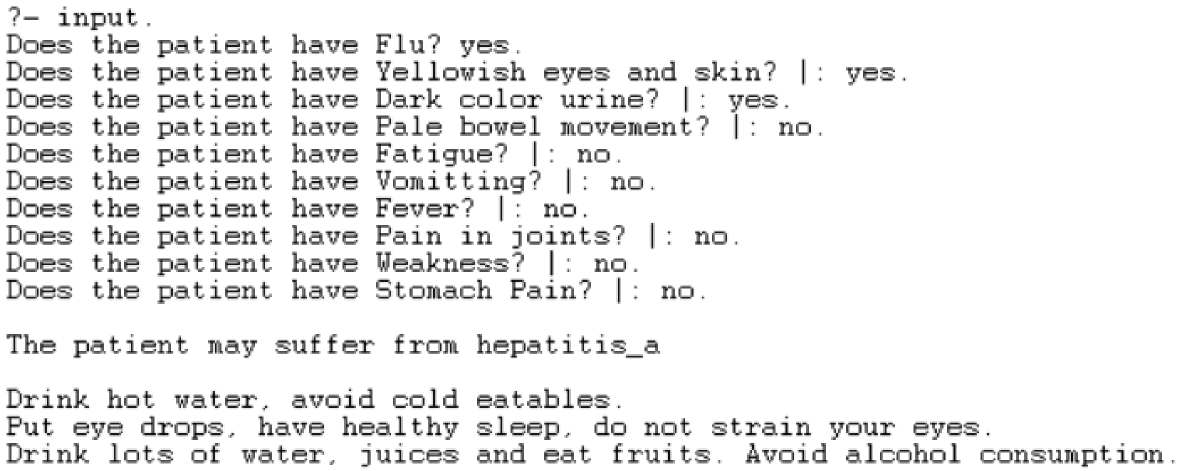
```
symptom('Flu').
symptom('Yellowish eyes and skin').
symptom('Dark color urine').
symptom('Pale bowel movement').
symptom('Fatigue').
symptom('Vomitting').
symptom('Fever').
symptom('Pain in joints').
symptom('Weakness').
symptom('Stomach Pain').

treatment('Flu', 'Drink hot water, avoid cold eatables.').
treatment('Yellowish eyes and skin', 'Put eye drops, have healthy sleep, do not strain your eyes.').
treatment('Dark color urine', 'Drink lots of water, juices and eat fruits. Avoid alcohol consumption.').
treatment('Pale bowel movement', 'Drink lots of water and exercise regularly.').
treatment('Fatigue', 'Drink lots of water, juices and eat fruits.').
treatment('Vomitting', 'Drink salt and water.').
treatment('Fever', 'Put hot water cloth on head and take crocin.').
treatment('Pain in Joints', 'Apply pain killer and take crocin.').
treatment('Weakness', 'Drink salt and water, eat fruits.').
treatment('Stomach Pain', 'Avoid outside food and eat fruits.').

input :- dynamic(patient/2),
   repeat,
   symptom(X),
   write('Does the patient have '),
   write(X),
   write('? '),
   read(Y),
   assert(patient(X,Y)),
   \+ not(X='Stomach Pain'),
   not(output).

disease(hemochromatosis) :-
   patient('Stomach Pain',yes),
   patient('Pain in joints',yes),
   patient('Weakness',yes),
   patient('Dark color urine',yes),
   patient('Yellowish eyes and skin',yes).

disease(hepatitis_c) :-
   not(disease(hemochromatosis)),
   patient('Pain in joints',yes),
   patient('Fever',yes),
   patient('Fatigue',yes),
   patient('Vomitting',yes),
   patient('Pale bowel movement',yes).

disease(hepatitis_b) :-
   not(disease(hemochromatosis)),
   not(disease(hepatitis_c)),
   patient('Pale bowel movement',yes),
   patient('Dark color urine',yes),
   patient('Yellowish eyes and skin',yes).

disease(hepatitis_a) :-
   not(disease(hemochromatosis)),
   not(disease(hepatitis_c)),
   not(disease(hepatitis_b)),
   patient('Flu',yes),
   patient('Yellowish eyes and skin',yes).
```

```
disease(jaundice) :-
   not(disease(hemochromatosis)),
   not(disease(hepatitis_c)),
   not(disease(hepatitis_b)),
   not(disease(hepatitis_a)),
   patient('Yellowish eyes and skin',yes).

disease(flu) :-
   not(disease(hemochromatosis)),
   not(disease(hepatitis_c)),
   not(disease(hepatitis_b)),
   not(disease(hepatitis_a)),
   patient('Flu',yes).
```

output:-

```
   nl,
   possible_diseases,
   nl,
   advice.

possible_diseases :- disease(X), write('The patient may suffer from '), write(X),nl.
advice :- symptom(X), patient(X,yes), treatment(X,Y), write(Y),nl, \+ not(X='Stomach Pain').
```

output

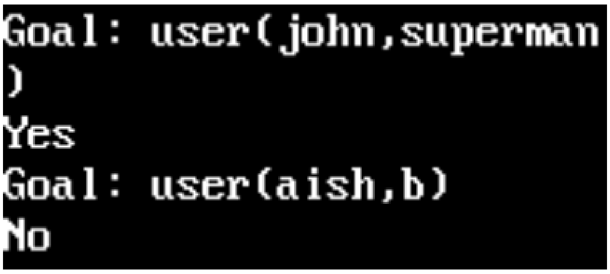


```
?- input.
Does the patient have Flu? yes.
Does the patient have Yellowish eyes and skin? |: yes.
Does the patient have Dark color urine? |: yes.
Does the patient have Pale bowel movement? |: no.
Does the patient have Fatigue? |: no.
Does the patient have Vomitting? |: no.
Does the patient have Fever? |: no.
Does the patient have Pain in joints? |: no.
Does the patient have Weakness? |: no.
Does the patient have Stomach Pain? |: no.

The patient may suffer from hepatitis_a

Drink hot water, avoid cold eatables.
Put eye drops, have healthy sleep, do not strain your eyes.
Drink lots of water, juices and eat fruits. Avoid alcohol consumption.
```

7)Write a program in prolog for password

```
domains
name,password=symbol
predicates
getinput(name,password)
logon
user(name,password)
clauses
logon:-
clearwindow,
getinput(_,_),
write("\n You are now Logged on."),nl.
logon:-
write("\n Sorry, You are not permitted access."),nl.
getinput(Name,Password):-
write("\n Please enter you Name: "),
readln(Name),nl,
write("\n Please enter you Password: "),
readln(Password),nl,
user(Name,Password).
user(john,superman).
user(aish,xyz).
```

Output:



```
Goal: user(john,superman
)
Yes
Goal: user(aish,b)
No
```

8)Write a program in C++/Prolog for DFS and BFS

Dfs

```
%connected(+Start, +Goal, -Weight)
connected(1,7,1).
connected(1,8,1).
connected(1,3,1).
connected(7,4,1).
connected(7,20,1).
connected(7,17,1).
connected(8,6,1).
connected(3,9,1).
connected(3,12,1).
connected(9,19,1).
connected(4,42,1).
connected(20,28,1).
connected(17,10,1).

connected2(X,Y,D) :- connected(X,Y,D).
connected2(X,Y,D) :- connected(Y,X,D).

next_node(Current, Next, Path) :-
   connected2(Current, Next, _),
   not(member(Next, Path)).
depth_first(Goal, Goal, _, [Goal]).
depth_first(Start, Goal, Visited, [Start|Path]) :-
   next_node(Start, Next_node, Visited),
   write(Visited), nl,
   depth_first(Next_node, Goal, [Next_node|Visited], Path).
```

Output

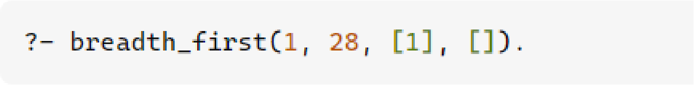
```
?- depth_first(1, 28, [1], P).
```

Bfs

```
%connected(+Start, +Goal, -Weight)
connected(1,7,1).
connected(1,8,1).
connected(1,3,1).
connected(7,4,1).
connected(7,20,1).
connected(7,17,1).
connected(8,6,1).
connected(3,9,1).
connected(3,12,1).
connected(9,19,1).
connected(4,42,1).
connected(20,28,1).
connected(17,10,1).

connected2(X,Y,D) :- connected(X,Y,D).
connected2(X,Y,D) :- connected(Y,X,D).
```

```
next_node(Current, Next, Path) :-
   connected2(Current, Next, _),
   not(member(Next, Path)).
breadth_first(Goal, Goal, _,[Goal]).
breadth_first(Start, Goal, Visited, Path) :-
   findall(X,
         (connected2(X,Start,_),not(member(X,Visited))),
         [T|Extend]),
   write(Visited), nl,
   append(Visited, [T|Extend], Visited2),
   append(Path, [T|Extend], [Next|Path2]),
   breadth_first(Next, Goal, Visited2, Path2).
```

Output



```
?- breadth_first(1, 28, [1], []).
```

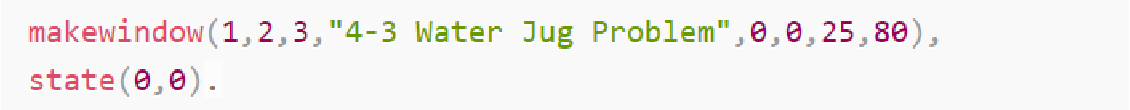
9)Write a program in C++/Prolog to implement Water Jug problem

```
/* Description:
"You are given two jugs, a 4-gallon one and a 3-gallon one. Neither have any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?".
*/
/* Production Rules:-
R1: (x,y) --> (4,y) if x < 4
R2: (x,y) --> (x,3) if y < 3
R3: (x,y) --> (x-d,y) if x > 0
R4: (x,y) --> (x,y-d) if y > 0
R5: (x,y) --> (0,y) if x > 0
R6: (x,y) --> (x,0) if y > 0
R7: (x,y) --> (4,y-(4-x)) if x+y >= 4 and y > 0
R8: (x,y) --> (x-(3-y),y) if x+y >= 3 and x > 0
R9: (x,y) --> (x+y,0) if x+y =< 4 and y > 0
R10: (x,y) --> (0,x+y) if x+y =< 3 and x > 0
*/

%database
   visited_state(integer,integer).

%predicates
   state(integer,integer).

%clauses
   state(2,0).

state(X,Y):- X < 4,
   not(visited_state(4,Y)),
   assert(visited_state(X,Y)),
   write("Fill the 4-Gallon Jug: (",X,",",Y,") --> (", 4,",",Y,")\n"),
   state(4,Y).

   state(X,Y):- Y < 3,
         not(visited_state(X,3)),
         assert(visited_state(X,Y)),
         write("Fill the 3-Gallon Jug: (", X,",",Y,") --> (", X,",",3,")\n"),
         state(X,3).

   state(X,Y):- X > 0,
         not(visited_state(0,Y)),
         assert(visited_state(X,Y)),
         write("Empty the 4-Gallon jug on ground: (", X,",",Y,") --> (", 0,",",Y,")\n"),
         state(0,Y).

   state(X,Y):- Y > 0,
         not(visited_state(X,0)),
         assert(visited_state(X,0)),
         write("Empty the 3-Gallon jug on ground: (", X,",",Y,") --> (", X,",",0,")\n"),
         state(X,0).

   state(X,Y):- X + Y >= 4,
         Y > 0,
         NEW_Y = Y - (4 - X),
         not(visited_state(4,NEW_Y)),
         assert(visited_state(X,Y)),
         write("Pour water from 3-Gallon jug to 4-gallon until it is full: (", X,",",Y,") --> (",
4,",",NEW_Y,")\n"),
         state(4,NEW_Y).
```

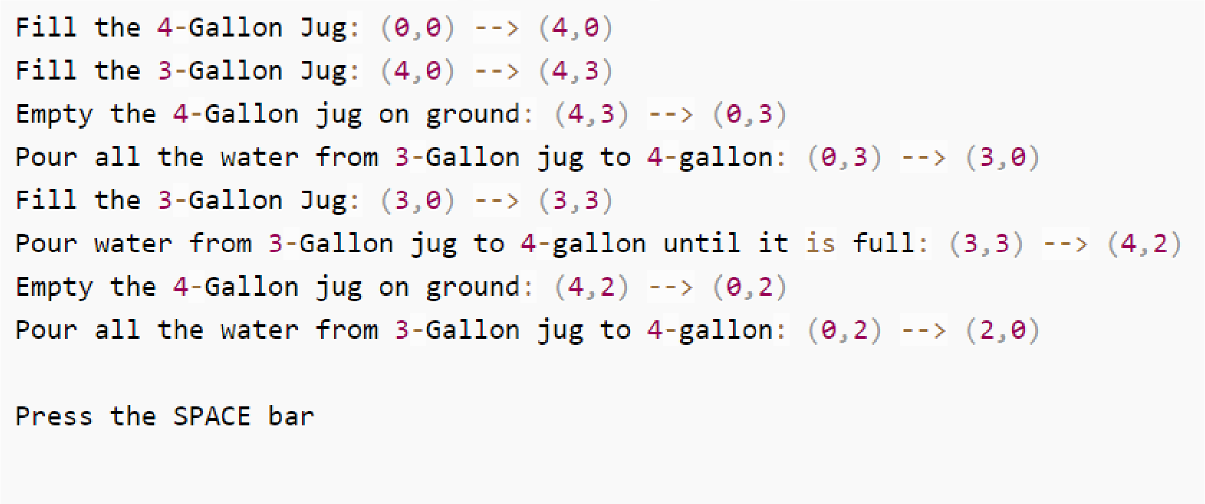
```
state(X,Y):- X + Y >=3,
    X > 0,
    NEW_X = X - (3 - Y),
    not(visited_state(X,3)),
    assert(visited_state(X,Y)),
    write("Pour water from 4-Gallon jug to 3-gallon until it is full: (", X,",",Y,") --> (", NEW_X,",",3,")\n"),
    state(NEW_X,3).

state(X,Y):- X + Y>=4,
    Y > 0,
    NEW_X = X + Y,
    not(visited_state(NEW_X,0)),
    assert(visited_state(X,Y)),
    write("Pour all the water from 3-Gallon jug to 4-gallon: (", X,",",Y,") --> (", NEW_X,",",0,")\n"),
    state(NEW_X,0).

state(X,Y):- X+Y >=3,
    X > 0,
    NEW_Y = X + Y,
    not(visited_state(0,NEW_Y)),
    assert(visited_state(X,Y)),
    write("Pour all the water from 4-Gallon jug to 3-gallon: (", X,",",Y,") --> (", 0,",",NEW_Y,")\n"),
    state(0,NEW_Y).

state(0,2):- not(visited_state(2,0)),
    assert(visited_state(0,2)),
    write("Pour 2 gallons from 3-Gallon jug to 4-gallon: (", 0,",",2,") --> (", 2,",",0,")\n"),
    state(2,0).
```

```
  state(2,Y):- not(visited_state(0,Y)),
     assert(visited_state(2,Y)),
     write("Empty 2 gallons from 4-Gallon jug on the ground: (", 2,",",Y,") --> (", 0,",",Y,")\n"),
     state(0,Y).


goal:-
      makewindow(1,2,3,"4-3 Water Jug Problem",0,0,25,80),
      state(0,0).
```

Output



```
makewindow(1,2,3,"4-3 Water Jug Problem",0,0,25,80),
state(0,0).
```



```
Fill the 4-Gallon Jug: (0,0) --> (4,0)
Fill the 3-Gallon Jug: (4,0) --> (4,3)
Empty the 4-Gallon jug on ground: (4,3) --> (0,3)
Pour all the water from 3-Gallon jug to 4-gallon: (0,3) --> (3,0)
Fill the 3-Gallon Jug: (3,0) --> (3,3)
Pour water from 3-Gallon jug to 4-gallon until it is full: (3,3) --> (4,2)
Empty the 4-Gallon jug on ground: (4,2) --> (0,2)
Pour all the water from 3-Gallon jug to 4-gallon: (0,2) --> (2,0)

Press the SPACE bar
```

10)Write a program in C++/Prolog to implement Tic tac toe Problem

```
play :- my_turn([]).
my_turn(Game) :-
   valid_moves(ValidMoves, Game, x),
   any_valid_moves(ValidMoves, Game).

any_valid_moves([], _) :-
   write('It is a tie'), nl.
any_valid_moves([_|_], Game) :-
   findall(NextMove, game_analysis(x, Game, NextMove), MyMoves),
   do_a_decision(MyMoves, Game).

% This can only fail in the beginning.
do_a_decision(MyMoves, Game) :-
   not(MyMoves = []),
   length(MyMoves, MaxMove),
   random(0, MaxMove, ChosenMove),
   nth0(ChosenMove, MyMoves, X),
   NextGame = [X | Game],
   print_game(NextGame),
   (victory_condition(x, NextGame) ->
      (write('I won. You lose.'), nl);
      your_turn(NextGame), !).

your_turn(Game) :-
   valid_moves(ValidMoves, Game, o),
   (ValidMoves = [] -> (write('It is a tie'), nl);
    (write('Available moves:'), write(ValidMoves), nl,
     ask_move(Y, ValidMoves),
     NextGame = [Y | Game],
    (victory_condition(o, NextGame) ->
      (write('I lose. You win.'), nl);
      my_turn(NextGame), !))).

ask_move(Move, ValidMoves) :-
   write('Give your move:'), nl,
   read(Move), member(Move, ValidMoves), !.

ask_move(Y, ValidMoves) :-
   write('not a move'), nl,
   ask_move(Y, ValidMoves).

movement_prompt(X, Y, ValidMoves) :-
   write('Give your X:'), nl, read(X), member(move(o, X, Y), ValidMoves), !,
   write('Give your Y:'), nl, read(Y), member(move(o, X, Y), ValidMoves).

% A routine for printing games.. Well you can use it.
print_game(Game) :-
   plot_row(0, Game), plot_row(1, Game), plot_row(2, Game).

plot_row(Y, Game) :-
   plot(Game, 0, Y), plot(Game, 1, Y), plot(Game, 2, Y), nl.

plot(Game, X, Y) :-
   (member(move(P, X, Y), Game), ground(P)) -> write(P) ; write('.').

% This system determines whether there's a perfect play available.
game_analysis(_, Game, _) :-
   victory_condition(Winner, Game),
   Winner = x. % We do not want to lose.
   % Winner = o. % We do not want to win. (egostroking mode).
```

```
    % true. % If you remove this constraint entirely, it may let you win.
game_analysis(Turn, Game, NextMove) :-
    not(victory_condition(_, Game)),
    game_analysis_continue(Turn, Game, NextMove).

game_analysis_continue(Turn, Game, NextMove) :-
    valid_moves(Moves, Game, Turn),
    game_analysis_search(Moves, Turn, Game, NextMove).

% Comment these away and the system refuses to play,
% because there are no ways to play this without a possibility of tie.
game_analysis_search([], o, _, _). % Tie on opponent's turn.
game_analysis_search([], x, _, _). % Tie on our turn.

game_analysis_search([X|Z], o, Game, NextMove) :- % Whatever opponent does,
    NextGame = [X | Game],                        % we desire not to lose.
    game_analysis_search(Z, o, Game, NextMove),
    game_analysis(x, NextGame, _), !.

game_analysis_search(Moves, x, Game, NextMove) :-
    game_analysis_search_x(Moves, Game, NextMove).

game_analysis_search_x([X|_], Game, X) :-
    NextGame = [X | Game],
    game_analysis(o, NextGame, _).
game_analysis_search_x([_|Z], Game, NextMove) :-
    game_analysis_search_x(Z, Game, NextMove).

% This thing describes all kinds of valid games.
valid_game(Turn, Game, LastGame, Result) :-
    victory_condition(Winner, Game) ->
```

```
        (Game = LastGame, Result = win(Winner)) ;
        valid_continuing_game(Turn, Game, LastGame, Result).

valid_continuing_game(Turn, Game, LastGame, Result) :-
    valid_moves(Moves, Game, Turn),
    tie_or_next_game(Moves, Turn, Game, LastGame, Result).

tie_or_next_game([], _, Game, Game, tie).
tie_or_next_game(Moves, Turn, Game, LastGame, Result) :-
    valid_gameplay_move(Moves, NextGame, Game),
    opponent(Turn, NextTurn),
    valid_game(NextTurn, NextGame, LastGame, Result).

% Victory conditions for tic tac toe.
victory(P, Game, Begin) :-
    valid_gameplay(Game, Begin),
    victory_condition(P, Game).

victory_condition(P, Game) :-
    (X = 0; X = 1; X = 2),
    member(move(P, X, 0), Game),
    member(move(P, X, 1), Game),
    member(move(P, X, 2), Game).

victory_condition(P, Game) :-
    (Y = 0; Y = 1; Y = 2),
    member(move(P, 0, Y), Game),
    member(move(P, 1, Y), Game),
    member(move(P, 2, Y), Game).

victory_condition(P, Game) :-
```

```
    member(move(P, 0, 2), Game),
    member(move(P, 1, 1), Game),
    member(move(P, 2, 0), Game).

victory_condition(P, Game) :-
    member(move(P, 0, 0), Game),
    member(move(P, 1, 1), Game),
    member(move(P, 2, 2), Game).

% This describes a valid form of gameplay.
% Which player did the move is disregarded.
valid_gameplay(Start, Start).

valid_gameplay(Game, Start) :-
    valid_gameplay(PreviousGame, Start),
    valid_moves(Moves, PreviousGame, _),
    valid_gameplay_move(Moves, Game, PreviousGame).

valid_gameplay_move([X|_], [X|PreviousGame], PreviousGame).
valid_gameplay_move([_|Z], Game, PreviousGame) :-
    valid_gameplay_move(Z, Game, PreviousGame).

% The set of valid moves must not be affected by the decision making
% of the prolog interpreter.
% Therefore we have to retrieve them like this.
% This is equivalent to the (∀x∈0..2)(∀y∈0..2)(....
% uh wait.. There's no way to represent this using those quantifiers.
valid_moves(Moves, Game, Turn) :-
    valid_moves_column(0, M1,    [], Game, Turn),
    valid_moves_column(1, M2,    M1, Game, Turn),
    valid_moves_column(2, Moves, M2, Game, Turn).
valid_moves_column(X, M3, M0, Game, Turn) :-
    valid_moves_cell(X, 0, M1, M0, Game, Turn),
    valid_moves_cell(X, 1, M2, M1, Game, Turn),
    valid_moves_cell(X, 2, M3, M2, Game, Turn).

valid_moves_cell(X, Y, M1, M0, Game, Turn) :-
    member(move(_, X, Y), Game) -> M0 = M1 ; M1 = [move(Turn,X,Y) | M0].

% valid_move(X, Y, Game) :-
%     (X = 0; X = 1; X = 2),
%     (Y = 0; Y = 1; Y = 2),
%     not(member(move(_, X, Y), Game)).

opponent(x, o).
opponent(o, x).
```

output

```
?- play.
...
x..
...
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,2),move(o,0,0)]
Give your move:
|: move(o, 1,1).
...
xo.
x..
Available moves:[move(o,2,2),move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0),move(o,0,0)]
```

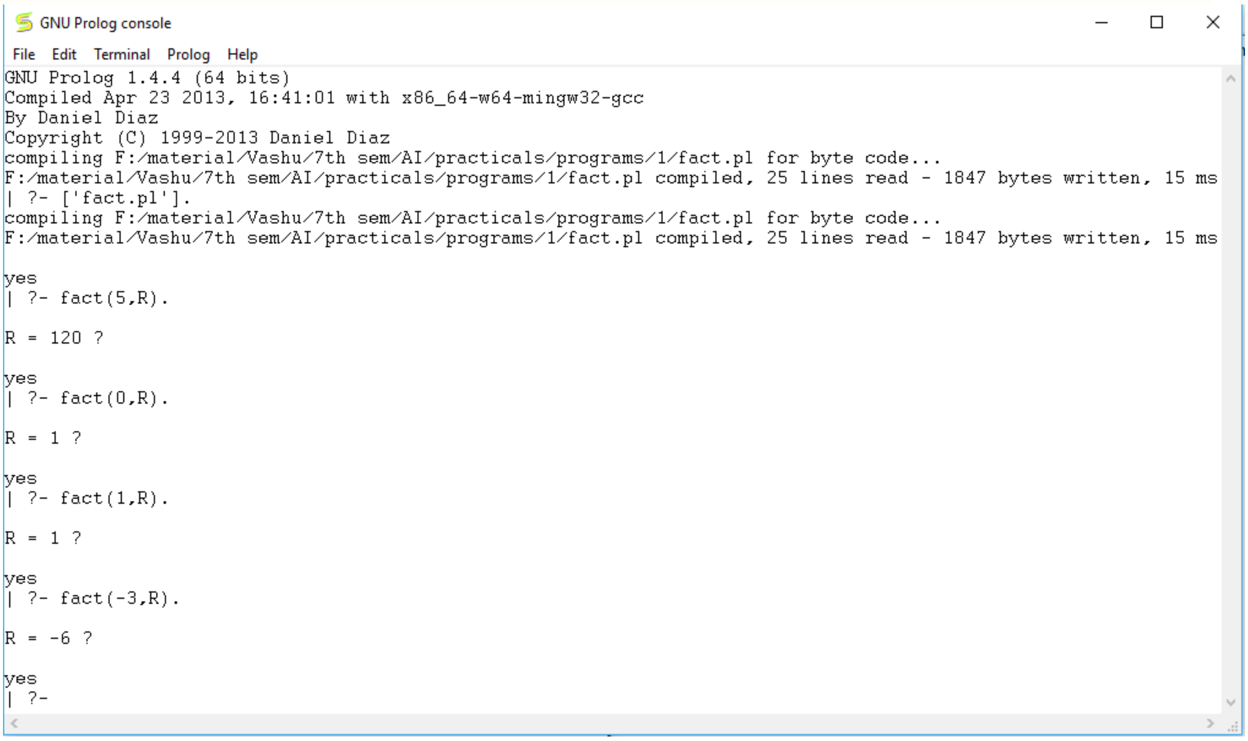
```
Give your move:
|: move(o, 0, 0).
o..
xo.
x.x
Available moves:[move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,0)]
Give your move:
|: move(o, 2, 0).
o.o
xo.
xxx
I won. You lose.
true.


?- play.
...
...
..x
Available moves:[move(o,2,1),move(o,2,0),move(o,1,2),move(o,1,1),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,1,1).
..x
.o.
..x
Available moves:[move(o,2,1),move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,1),move(o,0,0)]
Give your move:
|: move(o,2,1).
..x
xoo
```

```
..x
Available moves:[move(o,1,2),move(o,1,0),move(o,0,2),move(o,0,0)]
Give your move:
|: move(o,1,0).
.ox
xoo
.xx
Available moves:[move(o,0,2),move(o,0,0)]
Give your move:
|: move(o,0,2).
xox
xoo
oxx
It is a tie
true.
```

11)Write a program in prolog for factorial of given number

```
fact(0,1).
fact(N,F):-
(

 % The below is for +ve factorial.
 N>0 ->
  (
   N1 is N-1,
   fact(N1,F1),
   F is N*F1
  )
  ;

 % The below is for -ve factorial.
 N<0 ->
  (
   N1 is N+1,
   fact(N1,F1),
   F is N*F1
  )
).
```

Output



```
GNU Prolog 1.4.4 (64 bits)
Compiled Apr 23 2013, 16:41:01 with x86_64-w64-mingw32-gcc
By Daniel Diaz
Copyright (C) 1999-2013 Daniel Diaz
compiling F:/material/Vashu/7th sem/AI/practicals/programs/1/fact.pl for byte code...
F:/material/Vashu/7th sem/AI/practicals/programs/1/fact.pl compiled, 25 lines read - 1847 bytes written, 15 ms
| ?- ['fact.pl'].
compiling F:/material/Vashu/7th sem/AI/practicals/programs/1/fact.pl for byte code...
F:/material/Vashu/7th sem/AI/practicals/programs/1/fact.pl compiled, 25 lines read - 1847 bytes written, 15 ms

yes
| ?- fact(5,R).

R = 120 ?

yes
| ?- fact(0,R).

R = 1 ?

yes
| ?- fact(1,R).

R = 1 ?

yes
| ?- fact(-3,R).

R = -6 ?

yes
| ?-
```