Practical No.1

Title: Implementation of Logic programming

Aim: Understanding basics of prolog and implementation of prolog to solve tower of hanoi problem, water jug problem, tic-tac-toe problem and 8- Puzzle Problem.

Introduction:

What is Prolog :-

Prolog (programming in logic) is one of the most widely used programming languages in artificial intelligence research. As opposed to imperative languages such as C or Java (the latter of which also happens to be object-oriented) it is a declarative programming language. That means, when implementing the solution to a problem, instead of specifying how to achieve a certain goal in a certain situation, we specify what the situation (rules and facts) and the goal (query) are and let the Prolog interpreter derive the solution for us. Prolog is very useful in some problem areas, such as artificial intelligence, natural language processing, databases, . . . , but pretty useless in others, such as graphics or numerical algorithms.

Example : -

```
male(james1).

male(charles1).

male(charles2).

male(james2).

male(george1).
```

```
 female(catherine).

 female(elizabeth).

 female(sophia).

 parent(charles1, james1).

 parent(elizabeth, james1).

 parent(charles2, charles1).

 parent(catherine, charles1).

 parent(james2, charles1).

 parent(sophia, elizabeth).

 parent(george1, sophia).


mother(M,X):- parent(X,M),female(M),write(M),write(' is Mother of '),write(X),nl.


father(F,X):- parent(X,F),male(F).


sibling(S1,S2):- parent(S1,X), parent(S2,X).


grandfather(G,X) :- parent(Y,G),parent(X,Y).
```

Exercise -

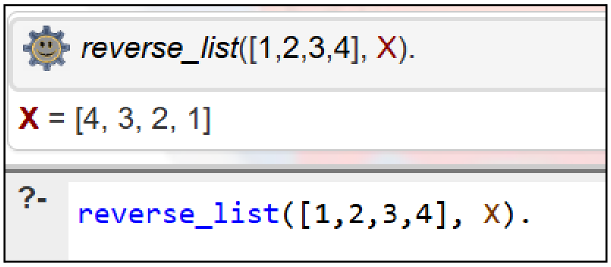
1. **Write a Prolog predicate reverse_list/2 that works like the built-in predicate reverse/2 (without using reverse/2).**

Implementation:

Program:

```
reverse_list([], []).

reverse_list([H|T], R) :-

   reverse_list(T, RT),

   append(RT, [H], R).
```

Output:



```
reverse_list([1,2,3,4], X).
X = [4, 3, 2, 1]

?- reverse_list([1,2,3,4], X).
```

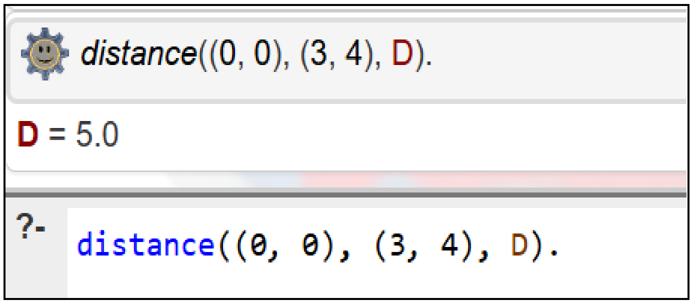
2. **Write a Prolog predicate distance/3 to calculate the distance between two points in the 2-dimensional plane. Points are given as pairs of coordinates.**

Implementation:

Program:

```
distance((X1, Y1), (X2, Y2), D) :-

 DX is X2 - X1,

 DY is Y2 - Y1,

D is sqrt(DX * DX + DY * DY).
```

OUTPUT:



```
distance((0, 0), (3, 4), D).

D = 5.0

?- distance((0, 0), (3, 4), D).
```

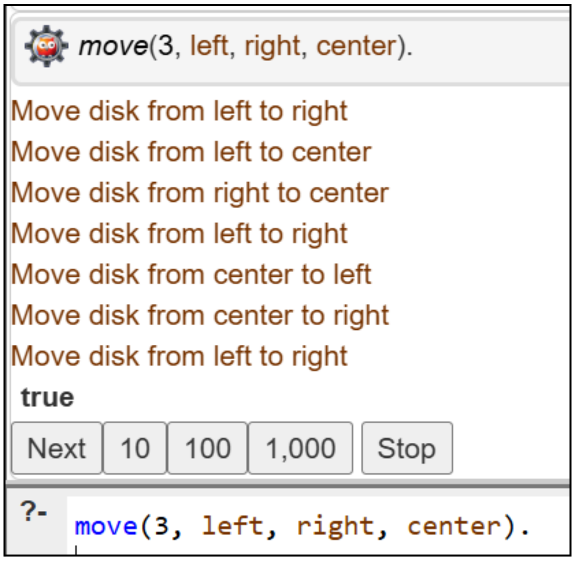
3. **Write a prolog program to solve Tower of hanoi Problem**

Implementation:

Program:

```
        move(1, Source, Destination, _) :-
  write('Move disk from '), write(Source), write(' to '), write(Destination), nl.
move(N, Source, Destination, Temp) :-
  N > 1,
  M is N - 1,
  move(M, Source, Temp, Destination),
  move(1, Source, Destination, _),
  move(M, Temp, Destination, Source).
```

OUTPUT:



```
move(3, left, right, center).
Move disk from left to right
Move disk from left to center
Move disk from right to center
Move disk from left to right
Move disk from center to left
Move disk from center to right
Move disk from left to right
true
?- move(3, left, right, center).
```

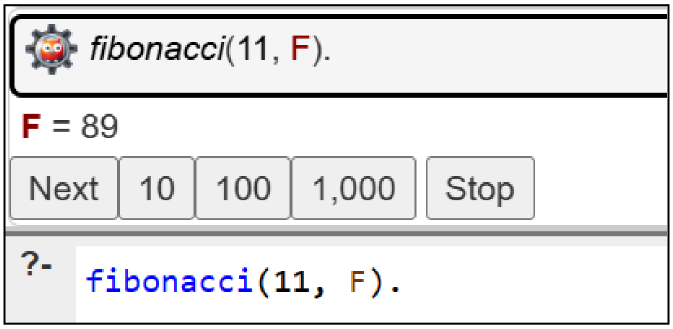
4. **Write a Prolog predicate fibonacci/2 to compute the nth Fibonacci number.**

Implementation:

Program:

```
fibonacci(0, 0).

fibonacci(1, 1).

fibonacci(N, F) :-

  N > 1,

  N1 is N - 1,

  N2 is N - 2,

  fibonacci(N1, F1),

  fibonacci(N2, F2),

  F is F1 + F2.
```

OUTPUT:



```
fibonacci(11, F).
F = 89
```

```
?- fibonacci(11, F).
```

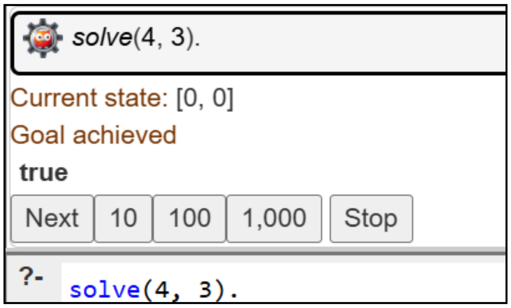
5. **Write a prolog program to solve water jug problem.**

Implementation:

Program:

```
solve(Jug4, Jug3) :- move([0, 0], Jug4, Jug3, []).

move([4, _], _, _, _) :- write('Goal achieved'), nl.

move([X, Y], Max4, Max3, Visited) :-

   member([X, Y], Visited) -> fail; % avoid loops

   write('Current state: '), write([X, Y]), nl,

   NextVisited = [[X, Y] | Visited],

   (

      move([4, Y], Max4, Max3, NextVisited);

      move([0, Y], Max4, Max3, NextVisited);

      move([X, 3], Max4, Max3, NextVisited);

      move([X, 0], Max4, Max3, NextVisited);

      Transfer is min(X, 3 - Y), NX is X - Transfer, NY is Y + Transfer, move([NX, NY], Max4, Max3,
NextVisited);

      Transfer is min(Y, 4 - X), NX is X + Transfer, NY is Y - Transfer, move([NX, NY], Max4, Max3,
NextVisited)

   ).
```

OUTPUT:



```
solve(4, 3).
Current state: [0, 0]
Goal achieved
true
```

```
?- solve(4, 3).
```

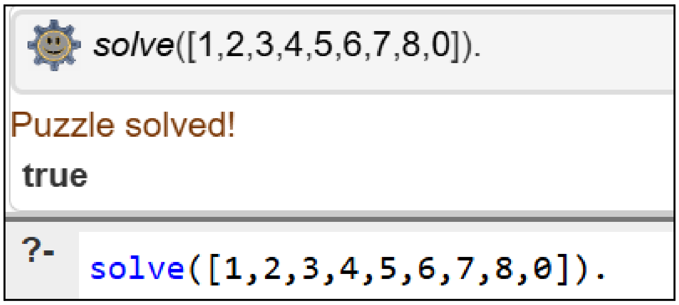
6. **Write a prolog program to solve 8- Puzzle Problem**

Implementation:

Program:

```
        goal([1,2,3,4,5,6,7,8,0]).

move([0,B,C,D,E,F,G,H,I], [B,0,C,D,E,F,G,H,I]).

% Define more moves...

solve(Puzzle) :- goal(Puzzle), write('Puzzle solved!').
```

OUTPUT:



```
solve([1,2,3,4,5,6,7,8,0]).
Puzzle solved!
true
?- solve([1,2,3,4,5,6,7,8,0]).
```