Kuldeep Srivastav

17CS25

Artificial Intelligence **(PROLOG Programming)**

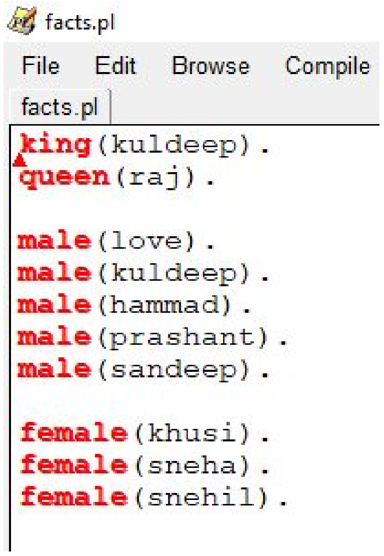
May 11, 2020

Prolog Programs

1.Write a program to implement simple facts and rules.

Program-(Facts)

```
king(kuldeep).
queen(raj).

male(love).
male(kuldeep).
male(hammad).
male(prashant).
male(sandeep).

female(khusi).
female(sneha).
female(snehil).
```



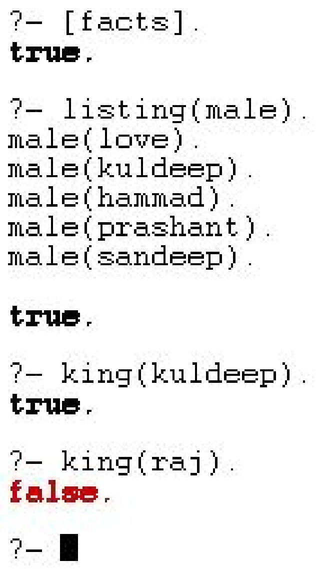
File Edit Browse Compile

facts.pl

```
king(kuldeep).
queen(raj).

male(love).
male(kuldeep).
male(hammad).
male(prashant).
male(sandeep).

female(khusi).
female(sneha).
female(snehil).
```

Output-

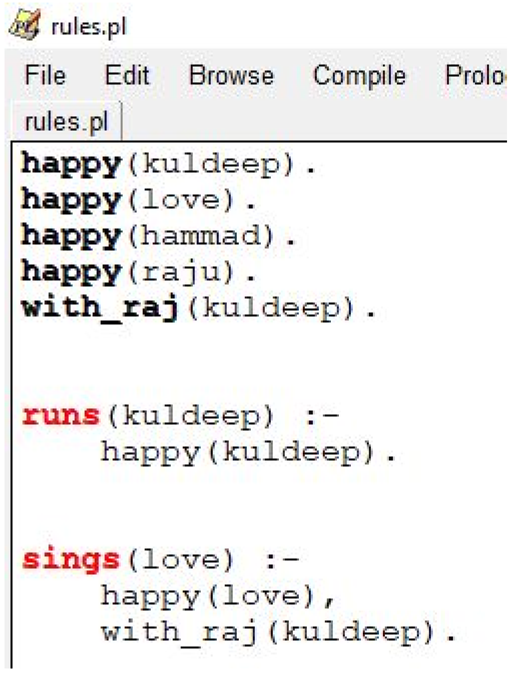
```
?- [facts].
true.

?- listing(male).
male(love).
male(kuldeep).
male(hammad).
male(prashant).
male(sandeep).
true.

?- king(kuldeep).
true.

?- king(raj).
false.
```



```
?- [facts].
true.

?- listing(male).
male(love).
male(kuldeep).
male(hammad).
male(prashant).
male(sandeep).

true.

?- king(kuldeep).
true.

?- king(raj).
false.

?-
```

Program-(Rules)

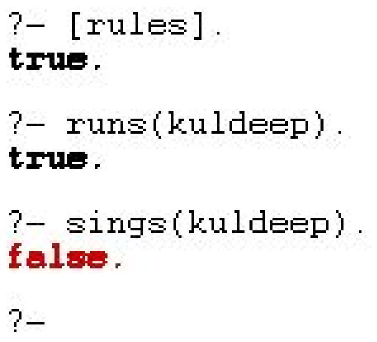
```
happy(kuldeep).
happy(love).
happy(hammad).
happy(raju).
with_raj(kuldeep).


runs(kuldeep) :-
   happy(kuldeep).


sings(love) :-
   happy(love),
   with_raj(kuldeep).
```



```
happy(kuldeep).
happy(love).
happy(hammad).
happy(raju).
with_raj(kuldeep).


runs(kuldeep) :-
    happy(kuldeep).


sings(love) :-
    happy(love),
    with_raj(kuldeep).
```

Output-

```
?- [rules].
true.

?- runs(kuldeep).
true.

?- sings(kuldeep).
false.
```

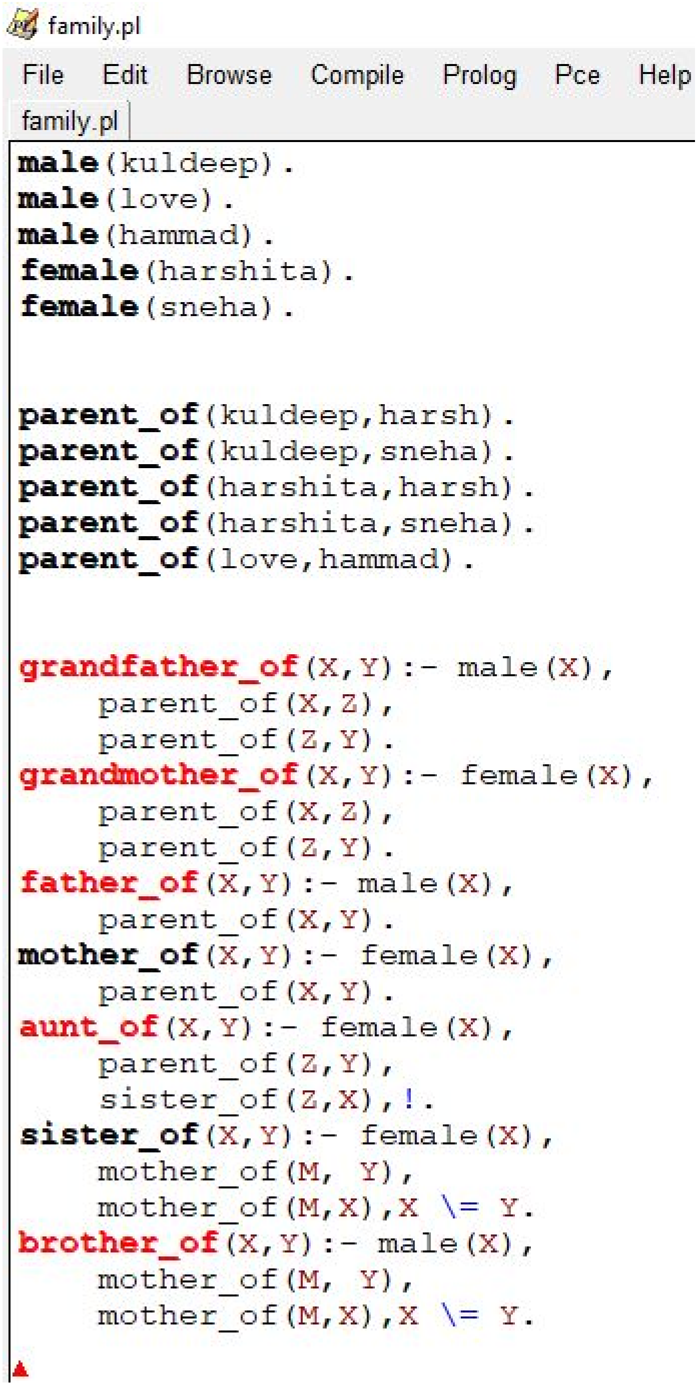


```
?- [rules].
true.

?- runs(kuldeep).
true.

?- sings(kuldeep).
false.

?-
```

2. Write a program to implement family tree.

Program-

```
male(kuldeep).
male(love).
male(hammad).
female(harshita).
female(sneha).

parent_of(kuldeep,harsh).
parent_of(kuldeep,sneha).
parent_of(harshita,harsh).
parent_of(harshita,sneha).
parent_of(love,hammad).

grandfather_of(X,Y):- male(X),
   parent_of(X,Z),
   parent_of(Z,Y).
grandmother_of(X,Y):- female(X),
   parent_of(X,Z),
   parent_of(Z,Y).
father_of(X,Y):- male(X),
   parent_of(X,Y).
mother_of(X,Y):- female(X),
   parent_of(X,Y).
aunt_of(X,Y):- female(X),
   parent_of(Z,Y),
   sister_of(Z,X),!.
sister_of(X,Y):- female(X),
   mother_of(M, Y),
   mother_of(M,X),X \= Y.
brother_of(X,Y):- male(X),
   mother_of(M, Y),
   mother_of(M,X),X \= Y.
```



```
male(kuldeep).
male(love).
male(hammad).
female(harshita).
female(sneha).


parent_of(kuldeep,harsh).
parent_of(kuldeep,sneha).
parent_of(harshita,harsh).
parent_of(harshita,sneha).
parent_of(love,hammad).


grandfather_of(X,Y):- male(X),
    parent_of(X,Z),
    parent_of(Z,Y).
grandmother_of(X,Y):- female(X),
    parent_of(X,Z),
    parent_of(Z,Y).
father_of(X,Y):- male(X),
    parent_of(X,Y).
mother_of(X,Y):- female(X),
    parent_of(X,Y).
aunt_of(X,Y):- female(X),
    parent_of(Z,Y),
    sister_of(Z,X),!.
sister_of(X,Y):- female(X),
    mother_of(M, Y),
    mother_of(M,X),X \= Y.
brother_of(X,Y):- male(X),
    mother_of(M, Y),
    mother_of(M,X),X \= Y.
```

Output-

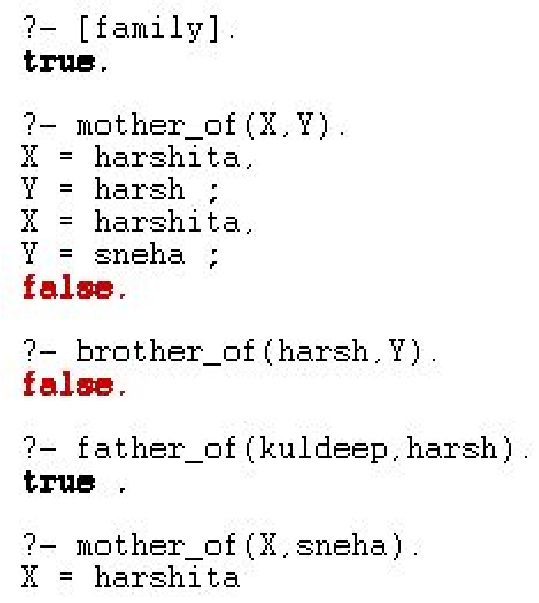
?- [family].
true.

?- mother_of(X,Y).
X = harshita,
Y = harsh ;
X = harshita,
Y = sneha ;
false.

?- brother_of(harsh,Y).
false.

?- father_of(kuldeep,harsh).
true

?- mother_of(X,sneha).
X = harshita

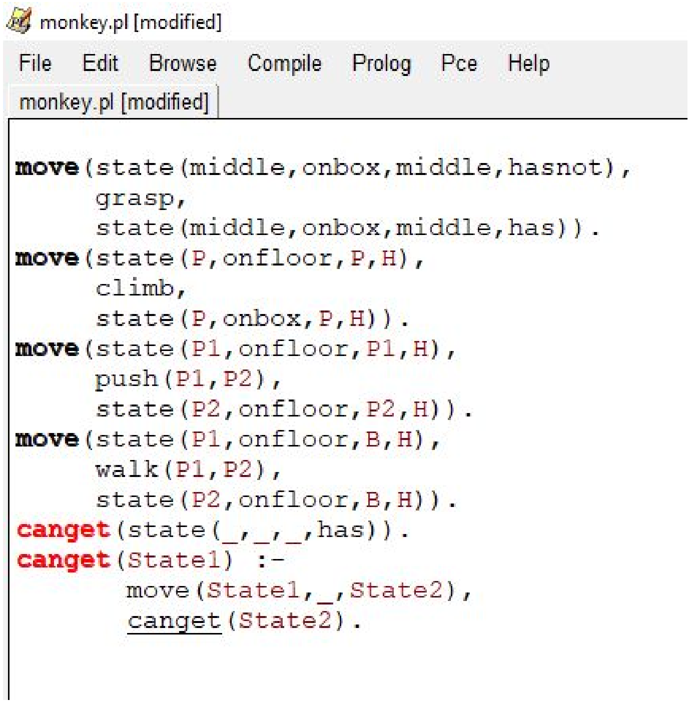


```
?- [family].
true.

?- mother_of(X,Y).
X = harshita,
Y = harsh ;
X = harshita,
Y = sneha ;
false.

?- brother_of(harsh,Y).
false.

?- father_of(kuldeep,harsh).
true .

?- mother_of(X,sneha).
X = harshita
```

3. Write a program to implement monkey banana problem using prolog.

Program-

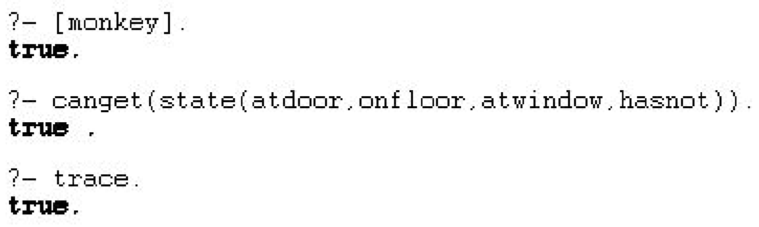
```
move(state(middle,onbox,middle,hasnot),
    grasp,
    state(middle,onbox,middle,has)).
move(state(P,onfloor,P,H),
    climb,
    state(P,onbox,P,H)).
move(state(P1,onfloor,P1,H),
    push(P1,P2),
    state(P2,onfloor,P2,H)).
move(state(P1,onfloor,B,H),
    walk(P1,P2),
    state(P2,onfloor,B,H)).
canget(state(_,_,_,has)).
canget(State1) :-
      move(State1,_,State2),
      canget(State2).
```



```
move(state(middle,onbox,middle,hasnot),
     grasp,
     state(middle,onbox,middle,has)).
move(state(P,onfloor,P,H),
     climb,
     state(P,onbox,P,H)).
move(state(P1,onfloor,P1,H),
     push(P1,P2),
     state(P2,onfloor,P2,H)).
move(state(P1,onfloor,B,H),
     walk(P1,P2),
     state(P2,onfloor,B,H)).
canget(state(_,_,_,has)).
canget(State1) :-
       move(State1,_,State2),
       canget(State2).
```

Output-

```
?- [monkey].
true.

?-
canget(state(atdoor,onfloor,atwindow,hasnot)).
true .

?- trace.
true.
```

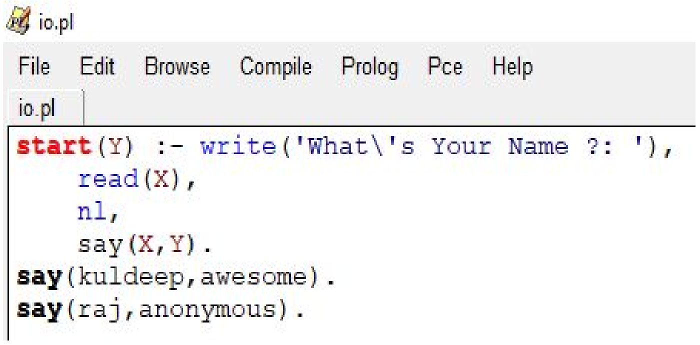


```
?- [monkey].
true.

?- canget(state(atdoor,onfloor,atwindow,hasnot)).
true .

?- trace.
true.
```

4. Write a program to implement I/O in prolog.

Program-

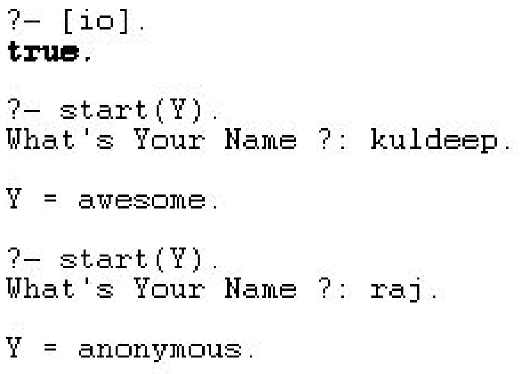
```
start(Y) :- write('What\'s Your Name ?: '),
   read(X),
   nl,
   say(X,Y).
say(kuldeep,awesome).
say(raj,anonymous).
```



```
start(Y) :- write('What\'s Your Name ?: '),
    read(X),
    nl,
    say(X,Y).
say(kuldeep,awesome).
say(raj,anonymous).
```

Output-

```
?- [io].
true.

?- start(Y).
What's Your Name ?: kuldeep.
Y = awesome.

?- start(Y).
What's Your Name ?: raj.
Y = anonymous.
```

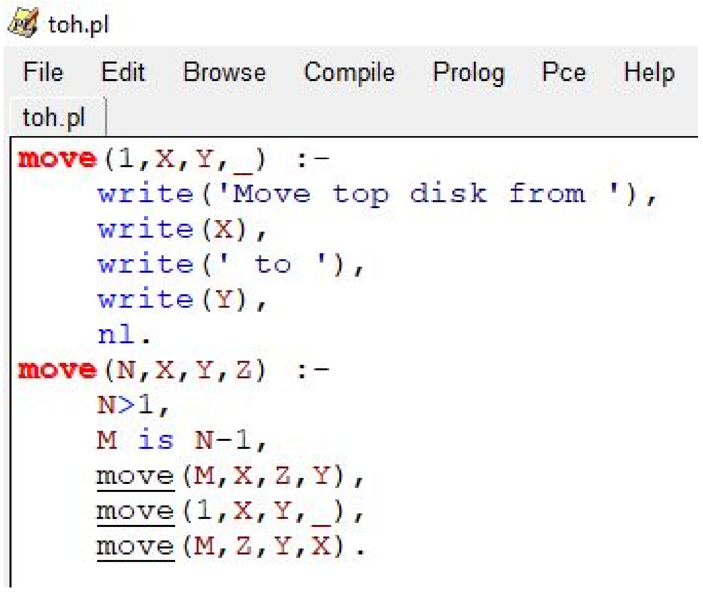


```
?- [io].
true.

?- start(Y).
What's Your Name ?: kuldeep.

Y = awesome.

?- start(Y).
What's Your Name ?: raj.

Y = anonymous.
```

5. Write a program to implement the Tower of Hanoi problem.

Program-

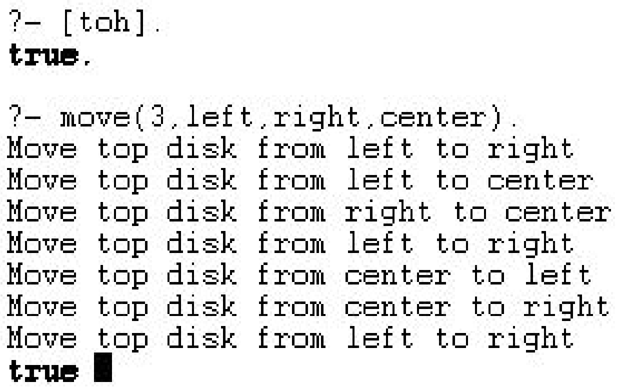
```
move(1,X,Y,_) :-
   write('Move top disk from '),
   write(X),
   write(' to '),
   write(Y),
   nl.
move(N,X,Y,Z) :-
   N>1,
   M is N-1,
   move(M,X,Z,Y),
   move(1,X,Y,_),
   move(M,Z,Y,X).
```



```
move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

Output-

```
?- [toh].
true.

?- move(3,left,right,center).

Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
true
```

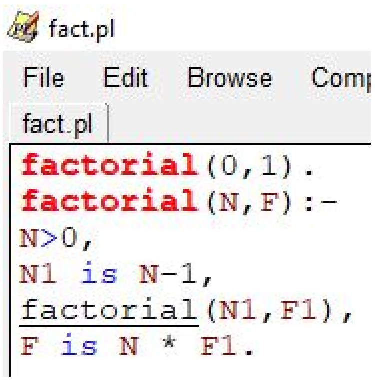


```
?- [toh].
true.

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

6. Write a program to find the factorial of a number using prolog.

Program: -

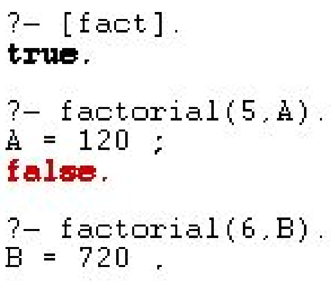
```
factorial(0,1).
factorial(N,F):-
N>0,
N1 is N-1,
factorial(N1,F1),
F is N * F1.
```



```
factorial(0,1).
factorial(N,F):-
N>0,
N1 is N-1,
factorial(N1,F1),
F is N * F1.
```

Output-

```
?- [fact].
true.

?- factorial(5,A).
A = 120 ;
false.

?- factorial(6,B).
B = 720 .
```

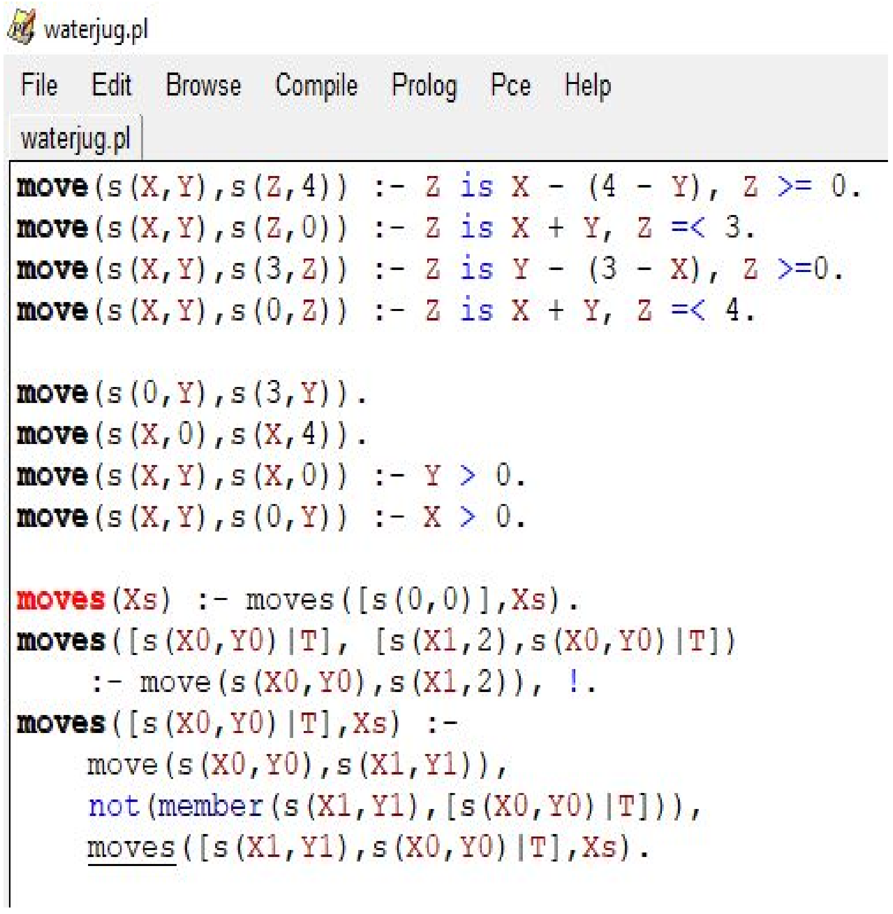


```
?- [fact].
true.

?- factorial(5,A).
A = 120 ;
false.

?- factorial(6,B).
B = 720 .
```

7. Write a program to implement water jug problem.

Program-

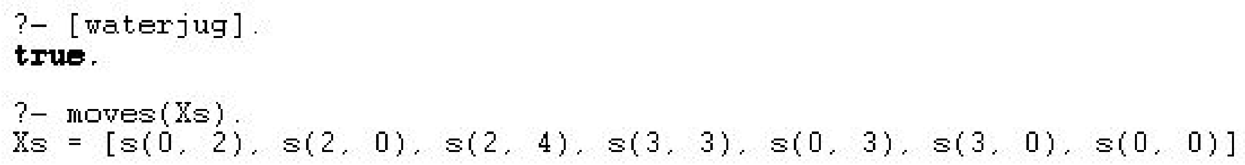
```
move(s(X,Y),s(Z,4)) :-

 Z is X - (4 - Y), Z >= 0.

move(s(X,Y),s(Z,0)) :-

Z is X + Y, Z =< 3.

move(s(X,Y),s(3,Z)) :-

Z is Y - (3 - X), Z >=0.

move(s(X,Y),s(0,Z)) :-

 Z is X + Y, Z =< 4.


move(s(0,Y),s(3,Y)).

move(s(X,0),s(X,4)).

move(s(X,Y),s(X,0)) :- Y > 0.

move(s(X,Y),s(0,Y)) :- X > 0.


moves(Xs) :- moves([s(0,0)],Xs).

moves([s(X0,Y0)|T], [s(X1,2),s(X0,Y0)|T])

   :- move(s(X0,Y0),s(X1,2)), !.

moves([s(X0,Y0)|T],Xs) :-

   move(s(X0,Y0),s(X1,Y1)),

   not(member(s(X1,Y1),[s(X0,Y0)|T])),

   moves([s(X1,Y1),s(X0,Y0)|T],Xs).
```



```
move(s(X,Y),s(Z,4)) :- Z is X - (4 - Y), Z >= 0.
move(s(X,Y),s(Z,0)) :- Z is X + Y, Z =< 3.
move(s(X,Y),s(3,Z)) :- Z is Y - (3 - X), Z >=0.
move(s(X,Y),s(0,Z)) :- Z is X + Y, Z =< 4.

move(s(0,Y),s(3,Y)).
move(s(X,0),s(X,4)).
move(s(X,Y),s(X,0)) :- Y > 0.
move(s(X,Y),s(0,Y)) :- X > 0.

moves(Xs) :- moves([s(0,0)],Xs).
moves([s(X0,Y0)|T], [s(X1,2),s(X0,Y0)|T])
    :- move(s(X0,Y0),s(X1,2)), !.
moves([s(X0,Y0)|T],Xs) :-
    move(s(X0,Y0),s(X1,Y1)),
    not(member(s(X1,Y1),[s(X0,Y0)|T])),
    moves([s(X1,Y1),s(X0,Y0)|T],Xs).
```

Output-

```
?- [waterjug].
true.

?- moves(Xs).
Xs = [s(0, 2), s(2, 0), s(2, 4), s(3, 3), s(0, 3), s(3, 0), s(0, 0)]
```



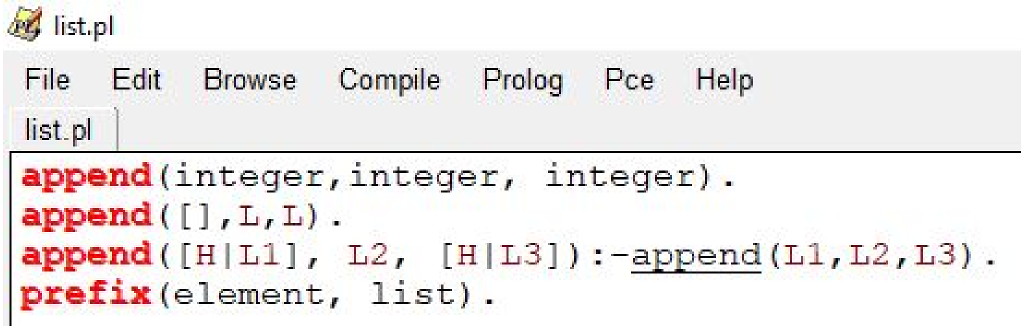
```
?- [waterjug].
true.

?- moves(Xs).
Xs = [s(0, 2), s(2, 0), s(2, 4), s(3, 3), s(0, 3), s(3, 0), s(0, 0)]
```

8. Write a program to implement various predicates on list:

- **i. Append**
- **ii. Prefix**

Program-

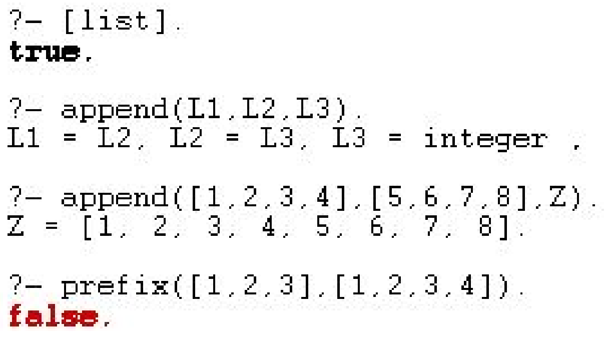
```
append(integer,integer, integer).
append([],L,L).
append([H|L1], L2, [H|L3]):-append(L1,L2,L3).
prefix(element, list).
```



```
append(integer,integer, integer).
append([],L,L).
append([H|L1], L2, [H|L3]):-append(L1,L2,L3).
prefix(element, list).
```

Output-

```
?- [list].
true.

?- append(L1,L2,L3).
L1 = L2, L2 = L3, L3 = integer .

?- append([1,2,3,4],[5,6,7,8],Z).
Z = [1, 2, 3, 4, 5, 6, 7, 8].

?- prefix([1,2,3],[1,2,3,4]).
false.
```

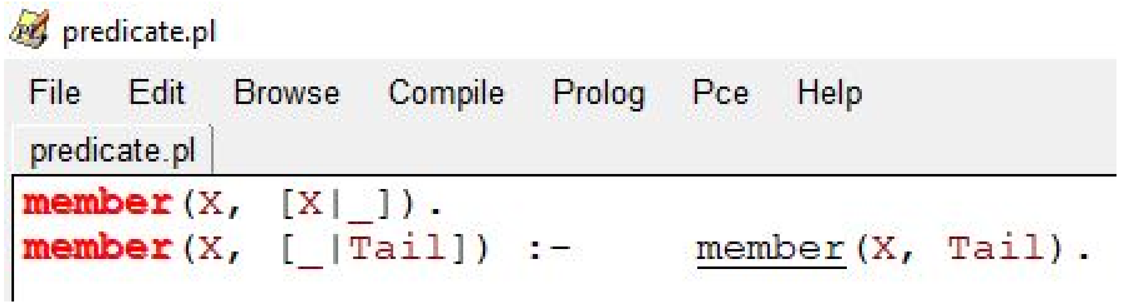


```
?- [list].
true.

?- append(L1,L2,L3).
L1 = L2, L2 = L3, L3 = integer .

?- append([1,2,3,4],[5,6,7,8],Z).
Z = [1, 2, 3, 4, 5, 6, 7, 8].

?- prefix([1,2,3],[1,2,3,4]).
false.
```

9. Write a program to implement member predicate on lists.

Program-

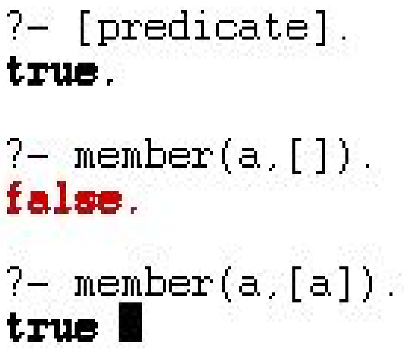
```
member(X, [X|_]).

member(X, [_|Tail]) :-     member(X, Tail).
```



```
member(X,  [X|_]).
member(X,  [_|Tail])  :-      member(X,  Tail).
```

Output-

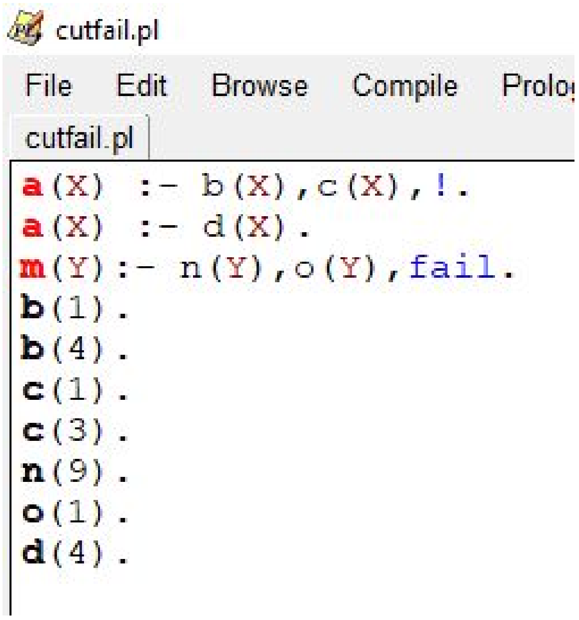
```
?- [predicate].
true.

?- member(a,[]).
false.

?- member(a,[a]).
true
```



```
?- [predicate].
true.

?- member(a,[]).
false.

?- member(a,[a]).
true
```

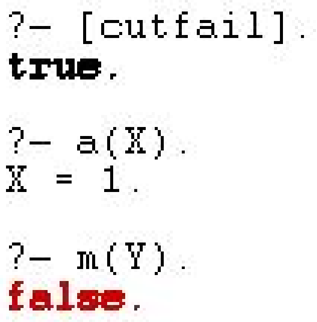
10. Write a program to implement cut and fail operations.

Program-



```
a(X) :- b(X),c(X),!.
a(X) :- d(X).
m(Y):- n(Y),o(Y),fail.
b(1).
b(4).
c(1).
c(3).
n(9).
o(1).
d(4).
```

Output-



```
?- [cutfail].
true.

?- a(X).
X = 1.

?- m(Y).
false.
```