Week-1: Write a prolog program to create knowledge base and implement the queries

Example 1:

Facts:

```
food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
dinner(pizza).
```

Rules:

```
meal(X) :- food(X).
```

Queries/Goals

```
?- food(pizza).   // Is pizza a food?
yes

?- meal(X),lunch(X).   // Which food  is meal and lunch?
X = sandwich ? ;
no

?- dinner(sandwich).  // Is sandwich a dinner?
no
```

Example 2:

Facts:

```
studies(charlie, csc135).
studies(olivia, csc135).
studies(jack, csc131).
studies(arthur, csc134).
teaches(kirke, csc135).
teaches(collins, csc131).
teaches(collins, csc171).
teaches(juniper, csc134).
```

Rules:

```
professor(X, Y) :- teaches(X, C), studies(Y, C).  // X is a professor of Y if X teaches C and Y studies C.
```

Queries/Goals:

```
?- studies(charlie, What).  // charlie studies What? Or What does charlies study?
What = csc135
yes

?- professor(kirke, Students). // Who are the students of professor kirke
Students = charlie ? ;
Students = olivia ? ;
no

?- studies(charlie, Which), teaches(Who,Which), write('charlie studies '), write(Which), write(' and
professor '), write(Who), write(' teaches '), write(Which).
```

charlie studies csc135 and professor kirke teaches csc135
Which = csc135
Who = kirke ?
yes

Example 3:

Facts:

owns(jack, car(bmw)).
owns(john, car(chevy)).
owns(olivia, car(civic)).
owns(jane, car(chevy)).
sedan(car(bmw)).
sedan(car(civic)).
truck(car(chevy)).

Goals/ Queries:

?- owns(john, X). // What does john own?
X = car(chevy)
yes

?- owns(john, _). // Does john own something?
true

?- owns(Who, car(chevy)). // Who owns car chevy?
Who = john ;
Who = jane .

?- owns(jane, X), sedan(X). // Does jane own sedan?
False

?- owns(jane, X), truck(X). // Does jane own truck?
X = car(chevy).

Example 4:

Facts & Rules:

female(pam).
female(liz).
female(pat).
female(ann).
male(jim).
male(bob).
male(tom).
male(peter).
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).

```
parent(bob,pat).
parent(pat,jim).
parent(bob,peter).
parent(peter,jim).
mother(X,Y):- parent(X,Y),female(X).
father(X,Y):- parent(X,Y),male(X).
haschild(X):- parent(X,_).
sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.
```

Implement the following Goals/Queries

```
?- parent(X,jim).
mother(X,Y).
haschild(X).
sister(X,Y).
```

Example 5:

Facts & Queries:

```
cat(fubby).
black_spots(fubby).
dog(figaro).
white_spots(figaro).
owns(mary, Pet):- cat(Pet), black_spots(Pet).  // mary owns a Pet if it is a cat and it has black spots
loves(Who, What):-owns(Who, What). // If someone owns something, he loves it.
```

Implement the following Queries/Goals:

```
// Who loves what?
// Mary owns something?
```

Example: 6

Facts & Rules

```
likes(harry,school).
likes(ron,broom).
likes(harry,X):-likes(ron,Y).
parent(sudip,puyus).
parent(sudip,raj).
male(piyus).
male(raj).
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),male(Y),format('~s is bother of ~s',[X,Y]).
```

Implement the following Queries/Goals:

```
% Find all brothers
% Find who likes harry
```

Week 2: Operators and Control Structures

1. Write a prolog program to ask and print your name?

```
name:- write('what is your name: '),nl,
        read(X),
        write('Hello  '),
        write(X).
```

Output:

```
?- name.
What is your name: Sree.
Hello  Sree
yes
```

2. Write a program to add 2 numbers?

```
add:- write('Enter the first number : '),nl,
      read(N1),nl,
      write('Enter the second number : '),
      read(N2), nl,
      Add is N1+N2,
      write(Add)
```

Output:

```
?- add.
Enter the first number : 2.
Enter the second number : 3.
5
Yes
```

3. Write a prolog program for simple calculator

```
calculator:- write('operations supported'),nl,
             write('1.Addition'),nl,
             write('2. Subtraction'),nl,
             write('3. Multiplication'),nl,
             write('4.Division'),nl,
             write('5.Modulo'),nl,nl,
             write('What do you want to do'),
             read(Choice),nl,
             write('Enter number 1'),
             read(N1),nl,
            write('Enter number 2'),
            read(N2),nl,
            cal(Choice,N1,N2).
cal(1,N1,N2):- Add is N1+N2, nl, write(Add).
cal(2,N1,N2):- Sub is N1-N2, nl, write(Sub).
cal(3,N1,N2):- Mul  is N1*N2, nl, write(Mul).
```

```
cal(4,N1,N2):- Div is N1/N2, nl, write(Div).
cal(5,N1,N2):- Mod is N1 mod N2, nl, write(Mod).
```

Output:

```
?- calculator.
operations supported
1.Addition
2. Subtraction
3. Multiplication
4.Division
5.Modulo

What do you want to do : 1.
Enter number 1  : 25.
Enter number 2 : 89.
114
yes
```

3. Write a prolog program to give an opportunity to the User to re enter the password 3 no. of times, on entering a wrong password.

```
db(ram,123).
db(sahil,888).
login:-login(0).
login(3):-write('max attempts crossed'),!.
login(N):-go(X,Y),db(X,Y),nl,write('login success').
login(N):-write('wrong'),nl,NN is N+1,login(NN).
go(X,Y):-write('Enter username'),read(X),write('Enter Password'),read(Y).
```

Output:

```
?- login.
Enter username : ace.
Enter Password : ace.
wrong
Enter username : sri.
Enter Password : rama.
wrong
Enter username : sri.
Enter Password : krishna.
wrong
max attempts crossed
(63 ms) yes
```

```
?- login.
Enter username : ram.
Enter Password : 123.
login success
true ?
yes
```

4. Write a prolog program to check if a given year is a leap Year or not?

```
go:-write('Enter the year'),nl,
      read(X),
      check(X).
check(X):-(N is (X mod 4),N>0,
                write('year is not an year');
                write('year is leap year')).
```

Output:

```
| ?- go.
Enter the year
2010.
year is not an year
true ?
(16 ms) yes

| ?- go.
Enter the year
2016.
year is leap year
yes
```

5. Write a prolog program to find a cube of a given number?

```
cube :-  write('Write a number: '),
           read(Number),
process(Number).
process(stop) :- !.
process(Number) :- C is Number * Number * Number,
                        format('cube of ~d : ~d ~n',[Number,C]),
                         cube.
```

Output:

```
?-  cube.
Write a number: 10.
cube of 10 : 1000
Write a number: 12.
cube of 12 : 1728
```

Write a number: 3.
cube of 3 : 27
Write a number: stop.

(47 ms) yes

6. Write a prolog program to find Min and Max of 2 numbers?

```
find_max(X, Y, X) :- X >= Y,!.
find_max(X, Y, Y) :- X < Y.

find_min(X, Y, X) :- X =< Y,!.
find_min(X, Y, Y) :- X > Y.
```

OR

```
max_find(X,Y,Max) :- X >= Y, !,Max = X; Max = Y.
```

Output:

```
?- max_find(23,45,Max).
Max = 45
yes
```

Week-3: Recursion, cut operator(!), "repeat" predicate

1. Write a prolog program to print values from 1 to 10

```
loop(10):-!.
loop(N):- N1 is N+1,
          write(N1),nl,
           loop(N1).
```

Output:

```
?- loop(0).
1
2
3
4
5
6
7
8
9
10
(31 ms) yes
```

2. Write a prolog program to print values from 10 to 1

```
loop(0):-!.
loop(N):- N1 is N-1,
          write(N1),nl,
          loop(N1).
```

Output:

```
?- loop(10).
  9
  8
  7
  6
  5
  4
  3
  2
  1
  0
(16 ms) yes
```

3. Write a prolog program to find factorial of a number using recursion.

```
factorial(0,1).
factorial(N,Res):-N>0,
                  N1 is N-1,
                  factorial(N1,R1),
                  Res is N*R1,
                  write(Res).
```

Output:

```
?- factorial(3,Result).
   Result = 6 ? ;
   no
```

4. **Write a prolog program to find the fibonacci number using recursion?**

```
fib(0,0).
fib(1,1).
fib(N,R):-  N>1,
            N1 is N-1, N2 is N-2,
            fib(N1,R1),fib(N2,R2),
            R is R1+R2.
```

Output:

```
?- fib(2,R).
R = 3
```

5. **Write a prolog program to find sum of integers in a given range A to B using recursion?**

```
sum_range(A,B,Result):- A=:=B, Result=A.
sum_range(A,B,Result):- A1 is A+1,
                        sum_range(A1,B,Result1),
                        Result is A + Result1.
```

Output:

```
?- sum_range(1,5,R).
   R = 15
(16 ms) yes
```

6. **Write a prolog program to find factorial with out recursion?**

```
factorial_itb(0,F,F).
factorial_itb(N,F,F):- N>0, T1 is T*N, N1 is N-1, factorial_itb(
```

7. **Write a prolog program to illustrate the use of cut(!) Operator?**

```
p(1).
p(2).
p(3).
p(4).
q(2).
q(3).
q(4).
r(5).
do:- p(N),q(N),write(N),r(N).
```

Output:

```
| ?- do.
234
no
```

```
p(1).
p(2).
p(3).
p(4).
q(2).
q(3).
q(4).
r(3).
do:- p(N),q(N),write(N),!,r(N).
```

Output:

```
| ?- do.
2
no
```

8. **Write a prolog program to illustrate the use of "repeat" Operator?**

```
name(sita).
name(ram).
go:- write('hi'),repeat,write('Enter the name:'),read(X),name(X).
```

Output:

```
| ?- go.
hiEnter the name:sree.
Enter the name:ace.
Enter the name:sita.
true ?
(15 ms) yes
```

Week 4: Lists, Towers of Hanoi problem

1. Built-in Predicates on Lists:

?- [1,2,3,4,5]=[Head|Tail].
Head = 1
Tail = [2,3,4,5]
(16 ms) yes

?- [1,2,3,4,5]=[_,X|_].
X = 2
Yes

?- [1]=[Head|Tail].
Head = 1
Tail = []
yes

?- length([1,2,3],A).
A = 3
yes

?- member(1,[1,2,3]).
true ?
yes

?- append([1],[1,2,3],A).
A = [1,1,2,3]
yes

| ?- reverse([1,2,3],A).
A = [3,2,1]
yes

?- select(1,[1,2,3],A).
A = [2,3] ?
yes

?-subtract([1],[1,2,3],A).
A = []
yes

2. Write a program to print the elements in the List?

```
writ([]).
writ([H|T]):- write(H),nl,writ(T).
```

Output:

```
?- writ([1,2,3,4,5]).
1
2
3
4
5
yes
```

3. Implement Towers of Hanoi in Prolog

Problem Statement: Towers of Hanoi Problem is a famous puzzle to move N disks from the source peg/tower to the target peg/tower using the intermediate peg as an auxiliary holding peg. There are two conditions that are to be followed while solving this problem –

- A larger disk cannot be placed on a smaller disk.
- Only one disk can be moved at a time.

The following diagram depicts the starting setup for N=3 disks.

To solve this, create the predicate move(N, Source, Target, auxiliary). Here N number of disks will have to be shifted from Source peg to Target peg keeping Auxiliary peg as intermediate.

Prolog Code:

```
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:

```
?- move(3,source,target,auxiliary).
Move top disk from source to target
```

Move top disk from source to auxiliary
Move top disk from target to auxiliary
Move top disk from source to target
Move top disk from auxiliary to source
Move top disk from auxiliary to target
Move top disk from source to target
true ?

Week 5: Monkey Banana Problem

1. Write a prolog program to solve Monkey Banana Problem

Problem Statement:
There is a monkey at the door into a room. In the middle of the room a banana is hanging from the ceiling. The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor. At the window of the room there is a box the monkey may use.
Understanding the problem:

There are 3 positions where the monkey and objects can be:
Atdoor – is the entrance of the room
Atwindow - is the place where box is placed
Atmiddle- is the place where bananas are hanging

Position of monkey and various objects:
1. Initially monkey is at door
2. Box is at window
3. Banana is at middle of the room hanging from the ceiling
4. Monkey can be either onfloor or onbox

Operations the monkey can perform:
The monkey can walk,push,climb and grasp

Define the predicate state as follows:
state(monkey_position,monkey_status,Box,Banana)
Where,
monkey_position defines position of the monkey
monkey_status defines onfloor or onbox condition

The variable Banana has 2 values: has and hasnot
The variable Box has position of the box in the room. It can be atdoor,atwindow, and atmiddle

Operation	Pre Condition	Post Condition
walk(L1,L2)	state(L1,onfloor,Box,Banana) % Monkey must be present at location L1 and must be onfloor%	state(L2,onfloor,Box,Banana) %Monkey changes its position to L2%
push(L1,L2)	state(L1,onfloor,L1,Banana) % Monkey and Box must be at position L1 so that can push it.%	state(L2,onfloor,L2,Banana) % Both Monkey and Box is at location L2%
climb	state(L,onfloor,L,Banana) % Mon_status is onfloor and both monkey and box at location L%	state(L,onbox,L,Banana) % mon_status is onbox at the same Location L
grasp	state(middle,onbox,middle,hasnot) % Both monkey and box are at the middle and monkey is on the box	state(middle,onbox,middle,has) % monkey now has banana

Predicates to solve the problem:

1. "move " predicate: Changes the state from one to another for the defined action
   - move(state(middle,onbox,middle,hasnot),grasp,state(middle,onbox,middle,has))
   - move(state(L,onfloor,L,Banana),climb,state(L,onbox,L,Banana))
   - move(state(L1,onfloor,L1,Banana),push(L1,L2), state(L2,onfloor,L2,,Banana)).
   - move(state(L1,onfloor,Box,Banana),walk(L1,L2), state(L2,onfloor,Box,Banana))
2. " canget" predicate: that returns true the moment Banana variable in state becomes has otherwise call move recursively

Prolog code:

```
% grab Banana
move(state(middle, onbox, middle, hasnot), grasp,   state(middle, onbox, middle, has)).

% climb box
move(state(L, onfloor, L, Banana), climb,   state(L, onbox, L, Banana)).

% push box from L1 and L2
move(state(L1, onfloor, L1, Banana), push(L1,L2),  state(L2, onfloor, L2, Banana)).


% walk from L1 and L2
move(state(L1, onfloor, Box, Banana), walk(L1, L2),  state(L2, onfloor, Box, Banana)).
```

```
%canget(state):monkey can get Banana in stage.
canget(state(_,_,_,has),[]).     %monkey already has it, goal state.
canget(State1,Plan):-move(State1,Action,State2),canget(State2,PartialPlan),add(Action,PartialPlan,Pla
n).

  %not goal state,  %do something(grab, climb, push, walk)  %canget from state2

add(X,L,[X|L]).        % add action to plan
```

Output:

```
?- canget(state(atdoor,onfloor,atdoor,hasnot),Plan).

Plan = [push(atdoor,middle),climb,grasp] ?
Yes

| ?- canget(state(middle,onbox,middle,hasnot),Plan).

Plan = [grasp] ?
Yes

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

Plan = [walk(atdoor,atwindow),push(atwindow,middle),climb,grasp] ?
(16 ms) yes
```

Week 6: Write a prolog program to solve Water Jug problem

Problem Statement:

Given 2 Jugs, a 4 liter one and a 3 liter one. Neither has any measuring mark on it. There is a pump that can be used to fill the Jugs with water. How can you get exactly 2 liter of water into 4 liter of the Jug.

State representation and initial state:
The State can be described as pair of integer [X,Y], where X represents the quantity of water in the 4 liter Jug. X={0,1,2,3,4} and Y represents the quantity of water in 3 liter Jug. Y={0,1,2,3}
Start State: (0,0)
Goal State:(2,0)

Production Rules:

Sno	Current State	Next State	Description
1	(X,Y) if X < 4	(4,Y)	Fill 4 Gallon Jug
2	(X,Y) if Y < 3	(X,3)	Fill 3 Gallon Jug
3	(X,Y) if X>0	(X-d,Y)	Pour some water out of 4 Gallon Jug
4	(X,Y) if Y>0	(X,Y-d)	Pour some water out of 3 Gallon Jug
5	(X,Y) if Y>0	(X,0)	Empty the 3 Gallon Jug
6	(X,Y) if X >0	(0,Y)	Empty the 4 Gallon Jug
7	(X,Y) if X+Y >= 4 and Y>0	(4,Y-(4-X))	Pour water from 3 Gallon Jug into 4 Gallon until 4 Gallon Jug is full
8	(X,Y) if X+Y >= 3 and X>0	(X-(3-Y),3)	Pour water from 4 Gallon Jug into 3 Gallon until 3 Gallon Jug is full
9	(X,Y) if X+Y <= 4 and Y>0	(X+Y,0)	Pour all water from Gallon 3 to Gallon 4
10	(X,Y) if X+Y <= 3 and X>0	(0,X+Y)	Pour all water from Gallon 4 to Gallon 3
11	(0,2)	(2,0)	Pour 2 liters from 3 Gallons Jug into 4 Gallons Jug
12	(2,Y)	(0,Y)	Empty the 2 liter in the 4 Gallon Jug on the ground

Solution to the water Jug Problem for the Initial State(0,3)

Sno	4 Liter Jug Contents	3 Liter Jug Contents	Rules applied
1	0 liter	0 liter	Initial State
2	0 liter	3 liter	Rule 2
3	3 Liter	0 Liter	Rule 9
4	3 Liter	3 Liter	Rule 2
5	4 Liter	2 Liter	Rule 7
6	0 Liter	2 Liter	Rule 5

7	2 Liter	0 Liter	Rule 9

Solution to the water Jug Problem for the Initial State(4,0)

Sno	4 Liter Jug Contents	3 Liter Jug Contents	Rules applied
1	0 liter	0 liter	Initial State
2	4 liter	0 liter	Rule 1
3	1 Liter	3 Liter	Rule 8
4	1 Liter	0 Liter	Rule 6
5	0 Liter	1 Liter	Rule 10
6	4 Liter	1 Liter	Rule 1
7	2 Liter	3 Liter	Rule 8
8	2 Liter	0 Liter	Rule 6

Prolog Code to solve Water Jug Problem:

```
fill(2,0):- nl,write('goal State reached').

fill(X,Y):- X=0,Y=<1,nl,
            write('Fill the 4 Gallon Jug :4,'), write(Y),
            fill(4,Y).

fill(X,Y):- Y=0,X=3,nl,
            write('Fill the 3 Gallon Jug:'),write('3'),
            fill(X,3).

fill(X,Y):-X+Y >= 4, Y=3,X=3,
            Y1 is Y-(4-X),nl,
            write('Pour water from 3 Gallon Jug to 4 Gallon Jug until it is full: 4, '), write(Y1),
            fill(4,Y1).

fill(X,Y):-X+Y >= 3, Y=<1,X=4,
           X1 is X-(3-Y),nl,
           write('Pour water from 4 Gallon Jug to 3 Gallon Jug until it is full:'), write(X1),write(',3'),
          fill(X1,3).

fill(X,Y):- X+Y =< 4, X=0,Y>=1,
            X1 is X+Y,nl,
            write('Pour all water from 3 Gallon Jug to 4 Gallon Jug:'),write(X1),write(',0'),
```

```
        fill(X1,0).

fill(X,Y):- X+Y =< 3, Y=0,
        Y1 is X+Y,nl,
        write('Pour all water from 4 Gallon Jug to 3 Gallon Jug:'),write(Y1),
        fill(0,Y1).

fill(X,Y):- Y=2,X=4,nl,
        write( 'Empty the 4 Gallon Jug on ground:0,'),write(Y),
        fill(0,Y).

fill(X,Y):- Y=3,X>=1,nl,
         write( 'Empty the 3 Gallon Jug on ground:'),write(X),write(',0'),
         fill(X,0).

fill(X,Y):- X>4, Y>3,write('4 Liter Jug Overflowed'),nl.

fill(X,Y):- X<4,Y>3,write('3 Liter Jug Overflowed'),nl.

fill(X,Y):- X>4,Y>3,write('Both 3 and 4  Liter Jug Overflowed'),nl.
```

Output:

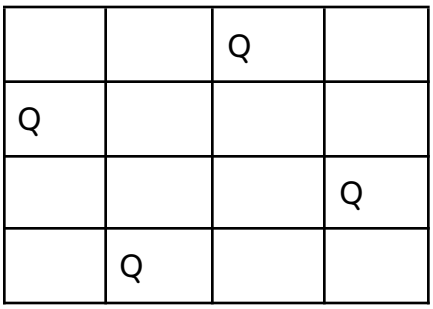
```
?- fill(0,3).

Pour all water from 3 Gallon Jug to 4 Gallon Jug:3,0
Fill the 3 Gallon Jug:3
Pour water from 3 Gallon Jug to 4 Gallon Jug until it is full: 4, 2
Empty the 4 Gallon Jug on ground:0,2
Pour all water from 3 Gallon Jug to 4 Gallon Jug:2,0
goal State reached

true ?

?- fill(4,0).

Pour water from 4 Gallon Jug to 3 Gallon Jug until it is full:1,3
Empty the 3 Gallon Jug on ground:1,0
Pour all water from 4 Gallon Jug to 3 Gallon Jug:1
Fill the 4 Gallon Jug :4,1
Pour water from 4 Gallon Jug to 3 Gallon Jug until it is full:2,3
Empty the 3 Gallon Jug on ground:2,0
goal State reached

true ?
```

Week 7:

1. Implement prolog program to solve 4 Queens Problem

Problem Statement:

The challenge is to set N queens on an N X N grid so that no queen can “take” any other queen. Queens can move horizontally, vertically and diagonal.



The solution to this puzzle can be represented as a special permutation of the list [1,2,3,4]. For example, The solution pictured above can be represented as [3,1,4,2], meaning that, in the first row place a queen in column 3, in the second row place a queen in column 1, etc.

Prolog program on 4-queens

```
% This program finds a solution to the 4 queens problem. That is, the problem of placing 4 queens on an
%4x4 chessboard so that no two queens attack each other.
% The prototype  board is passed in as a list with the rows instantiated from 1 to 4, and a corresponding
 % variable for each column.
% The Prolog program instantiates those column variables as it  finds the solution.

queens([]).  % when place queen in empty list, solution found
% otherwise, for each row
queens([ Row/Col | Rest]) :- queens(Rest),  % place a queen in each higher numbered row
                             member(Col, [1,2,3,4]), % pick one of the possible column positions
                             safe( Row/Col, Rest). % and see if that is a safe position
                                                   % if not, fail back  and try another column, until
                                                    % the columns are all tried, when fail back to
                                                    % previous row


safe(Anything, []).  % the empty board is always safe
% see if attack the queen in next row down
safe(Row/Col, [Row1/Col1 | Rest]) :- Col =\= Col1, % same column?
                                     Col1 - Col =\= Row1 - Row,% check diagonal
                                     Col1 - Col =\= Row - Row1,
                                     safe(Row/Col, Rest).% no attack on next row, try the rest of board

member(X, [X | Tail]).% member will pick successive column values
```

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

Output:

```
| ?- queens([1/C1, 2/C2, 3/C3, 4/C4]).

C1 = 3
C2 = 1
C3 = 4
C4 = 2 ? ;

C1 = 2
C2 = 4
C3 = 1
C4 = 3 ? ;

(94 ms) no
```

2. Write a prolog Program to find all paths from City A to City B.

```
road(birmingham,bristol, 9).
road(london,birmingham, 3).
road(london,bristol, 6).
road(london,plymouth, 5).
road(plymouth,london, 5).
road(portsmouth,london, 4).
road(portsmouth,plymouth, 8).

/*Here is the predicate we will call to find our paths, get_road/4. It basically calls the working predicate, that has two accumulators (one for the points already visited and one for the distance we went through).*/

get_road(Start, End, Visited, Result) :-  get_road(Start, End, [Start], 0, Visited, Result).

/*Here is the working predicate,
get_road/6 : get_road(+Start, +End, +Waypoints, +DistanceAcc, -Visited, -TotalDistance) :
The first clause tells that if there is a road between our first point and our last point, we can end here.*/

get_road(Start, End, Waypoints, DistanceAcc, Visited, TotalDistance) :-  road(Start, End, Distance),
                                                                          reverse([End|Waypoints], Visited),
                                                                          TotalDistance is DistanceAcc + Distance.

/*The second clause tells that if there is a road between our first point and an intermediate point, we can take it and then solve get_road(intermediate, end).*/

get_road(Start, End, Waypoints, DistanceAcc, Visited, TotalDistance) :-
   road(Start, Waypoint, Distance),
   \+ member(Waypoint, Waypoints),
   NewDistanceAcc is DistanceAcc + Distance,
```

```
get_road(Waypoint, End, [Waypoint|Waypoints], NewDistanceAcc, Visited, TotalDistance).
```

```
?-get_road(portsmouth, plymouth, Visited, Distance).

Distance = 8
Visited = [portsmouth,plymouth] ? ;

Distance = 9
Visited = [portsmouth,london,plymouth] ? ;

Distance = 18
Visited = [portsmouth,plymouth,london,plymouth] ? ;

(15 ms) no
```

—----------End----------