

# **CSL 303 : Artificial Intelligence**

## **TUTORIAL ASSIGNMENT 2**

Prolog Hands-On

**Date Assigned:** 19<sup>th</sup> August, 2021

**Date Submitted :** 23<sup>h</sup> August, 2021

**Submitted by:**

**Name:** S. Bhavyesh

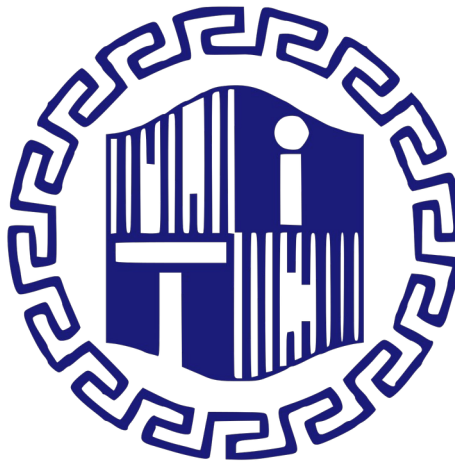
**Roll Number:** 191210045

**Branch:** Computer Science and Engineering

**Semester:** 5<sup>th</sup>

**Submitted to:** Dr. Chandra Prakash

**Department of Computer Science and Engineering**



**NATIONAL INSTITUTE OF TECHNOLOGY DELHI**

A-7, Institutional Area, near Satyawadi Raja Harish Chandra Hospital, New Delhi, Delhi 110040

## PART A: Exposition Problems

### 1. Classic Examples of Logical Programming

The three databases/programs were run to gain an exposure to application of Prolog at a large scale/ in developing AI systems.

- **Movie database**

```
7
8 Part 1: Write queries to answer the following questions.
9
10 a. In which year was the movie American Beauty released?
11 b. Find the movies released in the year 2000.
12 c. Find the movies released before 2000.
13 d. Find the movies released after 1990.
14 e. Find an actor who has appeared in more than one movie.
15 f. Find a director of a movie in which Scarlett Johansson appeared.
16 g. Find an actor who has also directed a movie.
17 h. Find an actor or actress who has also directed a movie.
18 i. Find the movie in which John Goodman and Jeff Bridges were co-stars.
19
20 Part 2: Add rules to the database to do the following,
21
22 a. released_after(M, Y) <- the movie was released after the given year.
23 b. released_before(M, Y) <- the movie was released before the given year.
24 c. same_year(M1, M2) <- the movies are released in the same year.
25 d. co_star(A1, A2) <- the actor/actress are in the same movie.
26
27 */
28
29 /** <examples> (Remove these if you want to give the exercises to students!)
30
31 ?- movie(american_beauty, Y).
32 ?- movie(M, 2000).
33 ?- movie(M, Y), Y < 2000.
34 ?- movie(M, Y), Y > 1999.
35 ?- actor(M1, A, _), actor(M2, A, _), M1 @> M2.
36 ?- actress(M, scarlett_johansson, _), director(M, D).
37 ?- actor( _ , A, _), director( _ , A).
```

```
58 movie(american_beauty, 1999).
59 director(american_beauty, sam_mendes).
60 actor(american_beauty, kevin_spacey, lester_burnham).
61 actress(american_beauty, annette_bening, carolyn_burnham).
62 actress(american_beauty, thora_birch, jane_burnham).
63 actor(american_beauty, wes_bentley, ricky_fitts).
64 actress(american_beauty, mena_suvari, angela_hayes).
65 actor(american_beauty, chris_cooper, col_frank_fitts_usmc).
66 actor(american_beauty, peter_gallagher, buddy_kane).
67 actress(american_beauty, allison_janney, barbara_fitts).
68 actor(american_beauty, scott_bakula, jim_olmeyer).
69 actor(american_beauty, sam_robards, jim_berkley).
70 actress(american_beauty, barry_del_sherman, brad_dupree).
71 actress(american_beauty, ara_celi, sale_house_woman_1).
72 actor(american_beauty, john_cho, sale_house_man_1).
73 actor(american_beauty, fort_atkinson, sale_house_man_2).
74 actress(american_beauty, sue_casey, sale_house_woman_2).
75 actor(american_beauty, kent_faulcon, sale_house_man_3).
76 actress(american_beauty, brenda_wehle, sale_house_woman_4).
77 actress(american_beauty, lisa_cloud, sale_house_woman_5).
78 actress(american_beauty, alison_faulk, spartanette_1).
79 actress(american_beauty, krista_goodsitt, spartanette_2).
80 actress(american_beauty, lily_houtkin, spartanette_3).
81 actress(american_beauty, carolina_lancaster, spartanette_4).
82 actress(american_beauty, romana_leah, spartanette_5).
83 actress(american_beauty, chekeshka_van_putten, spartanette_6).
84 actress(american_beauty, emily_zachary, spartanette_7).
85 actress(american_beauty, nancy_anderson, spartanette_8).
86 actress(american_beauty, reshma_gajjar, spartanette_9).
87 actress(american_beauty, stephanie_rizzo, spartanette_10).
88 actress(american_beauty, heather_joy_sher, playground_girl_1).
```

The following two simple queries were run:

- In which year was the movie 'American Beauty' released?
- Find the movies released in the year 2000.

The results are given below:

```
?- ['/home/bridges/Downloads/movies.pl'].
true.

?- movie(american_beauty,Y).
Y = 1999.

?- movie(M,2000).
M = down_from_the_mountain ;
M = o_brother_where_art_thou ;
M = ghost_world.

?- 
```

## OBSERVATION/COMMENTS

1. Although not shown here, Prolog supports both numeric and logical type of queries. For example, if we want to see which movies were released after 2000, the query will be:

*?- movie(M,Y),Y>2000.*

1<sup>st</sup> part is the logical part, 2<sup>nd</sup> one is numeric one.

2. One query might have multiple outputs. To display them all, press ; or Spacebar. If you want to stop the output flow, then type '.' after an output then hit enter.

- Eliza

```
5  eliza(Stimuli, Response) :-
6      template(InternalStimuli, InternalResponse),
7      match(InternalStimuli, Stimuli),
8      match(InternalResponse, Response),
9      !.
10
11  template([s([i,am]),s(X)], [s([why,are,you]),s(X),w('?')]).
12  template([w(i),s(X),w(you)], [s([why,do,you]),s(X),w(me),w('?')]).
13
14
15  match([],[]).
16  match([Item|Items],[Word|Words]) :-
17      match(Item, Items, Word, Words).
18
19  match(w(Word), Items, Word, Words) :-
20      match(Items, Words).
21  match(s([Word|Seg]), Items, Word, Words0) :-
22      append(Seg, Words1, Words0),
23      match(Items, Words1).
```

It is one of the oldest NLP programs that was written in the mid-1960s. Here, we ask the bot a question and in return we get a reply or a cross-question.

The queries that were executed were:

```
?- ['/home/bridges/Downloads/eliza.pl'].
true.

?- eliza([i,am,very,hungry],Response).
Response = [why, are, you, very, hungry, ?].

?- eliza([i,love,you],Response).
Response = [why, do, you, love, me, ?].

?- []
```

## OBSERVATION/COMMENTS

This simple example of a ‘bot’ illustrates that basic structure of languages can be ‘taught’ to the machine, by certain rules which dictate the subject-predicate placement in sentences.

- Expert system

```
6 prove(true) :- !.
7 prove((B, Bs)) :- !,
8   prove(B),
9   prove(Bs).
10 prove(H) :-
11   clause(H, B),
12   prove(B).
13 prove(H) :-
14   askable(H),
15   writeln(H),
16   read(Answer),
17   Answer == yes.
18
19
20 good_pet(X) :- bird(X), small(X).
21 good_pet(X) :- cuddly(X), yellow(X).
22
23 bird(X) :- has_feathers(X), tweets(X).
24
25 yellow(tweety).
26
27 askable(tweets(_)).
28 askable(small(_)).
29 askable(cuddly(_)).
30 askable(has_feathers(_)).
31
32
33 /** <examples>
34
35 ?- prove(good_pet(tweety)).
36
```

Expert systems are complex software/systems that are built to emulate human-based reasoning based on a large set of rules and facts related to a particular field – a knowledge-base.

In this example, we try to prove a statement. The expert system will ask certain questions in between and it accordingly continues with the proof.

Here, we prove that ‘tweety’ is a good pet.

```
?- ['/home/bridges/Downloads/expert_system.pl'].
true.

?- prove(good_pet(tweety)).
has_feathers(tweety)
|: yes.
tweets(tweety)
|: yes.
small(tweety)
|: yes.

true .

?-
```

## OBSERVATION/COMMENTS

This example of expert system tells us that it is not necessary to add every fact into the knowledge base of the program. It gives us a way to interpret rules and ask for missing knowledge.

## PART B : Conceptual Questions

### 2. Run the sample example given.

#### a) Discuss the sample example given of Sam's likes and dislikes in food.

The given example shows what foods Sam likes or dislikes. His preference is largely driven by the category of the food. Foods that are either mild Indian, Chinese, or Italian are preferred by Sam, along with chips. Rest of the foods that don't fit any of the criteria, are disliked by Sam.

#### b) Identify the facts in the sample example.

- Sam likes chips.
- Curry is Indian.
- Dahl is Indian.
- Tandoori is Indian.
- Kurma is Indian.
- Dahl is mild.
- Tandoori is mild.
- Kurma is mild.
- Chow Mein is Chinese.
- Chop Suey is Chinese.
- Sweet and Sour is Chinese.
- Pizza is Italian.
- Spaghetti is Italian.

#### c) Identify the rules in the taken example.

- Sam likes a food item if it is Indian and mild.
- Sam likes a food item if it is Chinese.
- Sam likes a food item if it is Italian.

#### d) Run what does Sam likes.

```
?- ['/home/bridges/Downloads/likes.pl'].
true.

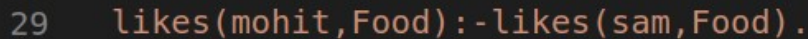
?- likes(sam,X).
X = dahl ;
X = tandoori ;
X = kurma ;
X = chow_mein ;
X = chop_suey ;
X = sweet_and_sour ;
X = pizza ;
X = spaghetti ;
X = chips.
```

e) Sam likes curry.



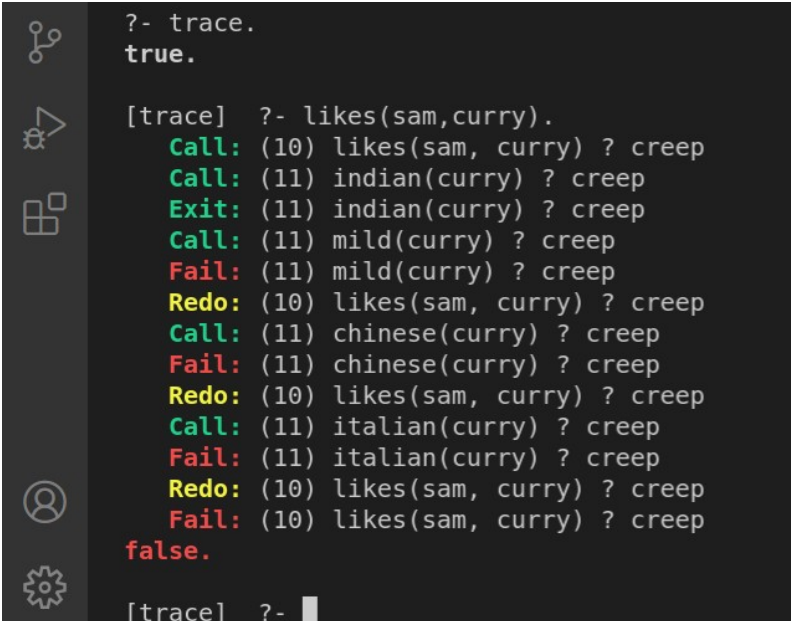
```
?- likes(sam,curry).  
false.
```

f) Add a new rule that Mohit likes whatever Sam likes.



```
29 likes(mohit,Food):-likes(sam,Food).
```

g) Tracing the execution of a Prolog query allows you to see all of the goals that are executed as part of the query, in sequence, along with whether or not they succeed. Show the steps occur in the above program.



```
?- trace.  
true.  
  
[trace] ?- likes(sam,curry).  
  Call: (10) likes(sam, curry) ? creep  
  Call: (11) indian(curry) ? creep  
  Exit: (11) indian(curry) ? creep  
  Call: (11) mild(curry) ? creep  
  Fail: (11) mild(curry) ? creep  
  Redo: (10) likes(sam, curry) ? creep  
  Call: (11) chinese(curry) ? creep  
  Fail: (11) chinese(curry) ? creep  
  Redo: (10) likes(sam, curry) ? creep  
  Call: (11) italian(curry) ? creep  
  Fail: (11) italian(curry) ? creep  
  Redo: (10) likes(sam, curry) ? creep  
  Fail: (10) likes(sam, curry) ? creep  
false.  
[trace] ?- 
```

## OBSERVATION/COMMENTS

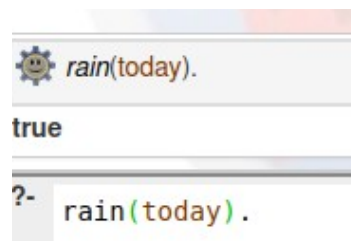
The trace command shows us how a PROLOG program uses backtracking in order to achieve its mini-goals within the program, as well as to come up with an answer to our query.

### 3. Consider the following Knowledge Base:

*The humidity is high or the sky is cloudy.  
If the sky is cloudy, then it will rain.  
If the humidity is high, then it is hot.  
It is not hot today.*

**Query :** *Will it rain today ?*

```
1 humidity(X,high):-not(sky(X,cloudy)).
2 sky(X,cloudy):-not(humidity(X,high)).
3
4 not(hot(today)).
5 not(humidity(X,high)):-not(hot(X)).
6 not(sky(X,cloudy)):-not(rain(X)).
7
8 hot(X):-humidity(X,high).
9 rain(X):-sky(X,cloudy).
10
```



### OBSERVATION/COMMENTS

1. This exercise brings to light a major flaw in PROLOG, that is, it does not provide any direct support for disjunctive(OR) statements.

The first two lines are representing the 1<sup>st</sup> statement in the problem.

2. Lines 5,6 are contrapositives of statements 8 and 9, since PROLOG is more suited for inferences that allow the application of *modus ponens* ( $q \leftarrow p$ ,  $p$  therefore  $q$ ).



4. Write a prolog program to calculate the sum of two numbers.

5. Write a Prolog program to implement max(X, Y, Max) so that Max is the maximum of two numbers X and Y.

6. Write a program in PROLOG to implement factorial (N, Fac) where Fac represents the factorial of a number N.

```
1  % Question 4
2  sum(X,Y,Z):-Z is X+Y.
3
4  % Question 5
5  max(X,Y,Z):-Z is max(X,Y).
6
7  % Question 6
8  factorial(0,1).
9  factorial(N,X):-
10     N>0,
11     Num is N-1,
12     factorial(Num,Y),
13     X is N*Y.
```

```
bridges@bridges-CF-C2AHCCZC7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/home/bridges/TA_2_Prob_4_5_6.pl'].
true.

?- sum(7,11,Z).
Z = 18.

?- max(10,20,Z).
Z = 20.

?- factorial(5,Z).
Z = 120 .

?-
```

## OBSERVATION/COMMENTS

1. To find the sum we simply assign the numeric sum to a variable:  $Z$  is  $X+Y$ , where  $X, Y$  are real numbers.
2. To find the max value of the two numbers, we simply use the in-built  $\max()$  function:  $Z$  is  $\max(X,Y)$ , where  $X,Y$  are real numbers.
3. To find the factorial of a number we define a recursive procedure  $\text{factorial}(N,Z)$  such that

```
factorial(0,1).  
factorial(N,X):-  
    N>0,  
    Num is N-1,  
    factorial(Num,Y),  
    X is N*Y.
```

$\text{factorial}(0,1)$  defines the base case of the recursion('factorial of 0 is 1'). To understand what happens in the second part, let us see through an example:

To find:  $\text{factorial}(2,X)$ .

```
factorial(2,X):-  
    2>0,  
    Num is 2-1,  
    factorial(1,Y),  
    X is 2*Y.
```

```
factorial(1,Y):-  
    1>0,  
    Num is 1-1,  
    factorial(0,Z),  
    Y is 1*Z.
```

```
factorial(0,Z).
```

$Z=1$  satisfies the above constraint. Going back up,

```
factorial(1,1):-  
    1>0,  
    Num is 1,  
    factorial(0,1),  
    Y is 1*1.
```

```
factorial(2,2):-  
    2>0,  
    Num is 2-1,  
    factorial(1,1),  
    Z is 2*1.
```

So, factorial of 2 is 2.

4. Another way to find the max element is

$$\begin{aligned} \text{max}(X,Y,Z):-Y>X,Z \text{ is } Y. \\ \text{max}(X,Y,Z):-X>=Y, Z \text{ is } X. \end{aligned}$$

5. The program snippet for factorial uses recursion, which is a cornerstone for programming in PROLOG, as we will see in the case of lists and their manipulations.

6. If, instead of *is*, we use *=*, then we get the following:

```
bridges@bridges-CF-C2AHCCZC7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- Z is 4+5.
Z = 9.

?- Z = 4+5.
Z = 4+5.

?- 
```

7. Write a Prolog program to implement `multi(N1, N2, Res)` : where N1 and N2 denotes the numbers to be multiplied and Res represents the result.

8. Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] where edge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a program to check whether there is a route from one node to another node.

9. Write a Prolog program to implement `memb(X, L)`: to check whether X is a member of L or not.

```
1  % Question 7
2  multi(N1,N2,Res):-Res is N1*N2.
3
4  % Question 8
5  edge(p,q).
6  edge(q,r).
7  edge(q,r).
8  edge(q,s).
9  edge(s,t).
10
11 % Question 9
12 route(X,Y):-edge(X,Y).
13 route(X,Y):-edge(X,Z),route(Z,Y).
14
15 member(X,[X|Tail]).
16 member(X,[Head|Tail]):-member(X,Tail).
```

```
bridges@bridges-CF-C2AHCCZC7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/home/bridges/TA_2_Prob_7_8_9.pl'].
Warning: /home/bridges/TA_2_Prob_7_8_9.pl:15:
Warning: Singleton variables: [Tail]
Warning: /home/bridges/TA_2_Prob_7_8_9.pl:16:
Warning: Singleton variables: [Head]
true.

?- multi(5,9,Z).
Z = 45.

?- route(p,s).
true .

?- member(4,[1,2,3,4,5]).
true .
```

## OBSERVATION/COMMENTS

1. To find the product of two numbers we simply assign the value to a variable:  $Z$  is  $X*Y$ , where  $X$ ,  $Y$  are real numbers.

2. To check for path existence between nodes  $X$  and  $Y$ , we define a recursive procedure which will handle two cases:

- case 1: edge exists between  $X$  and  $Y$ .
- case 2: edge does not exist between  $X$  and  $Y$ .

In case 2, the program tries to find a vertex  $Z$  such that there exists an edge between  $X$  and  $Z$ , and a path exists between  $Z$  and  $Y$ . Then, it recurses until the condition is met.

3. To check for list membership, there are 2 cases:

- case 1:  $X$  is head of list  $L$ .
- case 2:  $X$  is a member of tail of list  $L$ .

Case 2 is the recursion step, where the program recurses to the end of the list or where case 1 is satisfied for the concerned sublist.

4. Lists can only be accessed by their head.

10. Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.

```
bridges@bridges-CF-C
GNU nano 4.8 TA 2 Prob :
concat([],L,L).
concat([X|L1],L2,[X|L3]):-concat(L1,L2,L3).

```

```
bridges@bridges-CF-C2AHCCZ7: ~
bridges@bridges-CF-C2AHCCZ7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/home/bridges/TA_2_Prob_10.pl'].
true.

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

?- 
```

## OBSERVATION/COMMENTS

1. Two cases emerge in concatenation of lists:

- case 1: 1<sup>st</sup> list is empty.
- case 2: 1<sup>st</sup> list is non-empty.

2. In case 1, we simply assign list L3, the resultant list, the value of list L2.

3. In case 2, we separate the head of list L1 from the list and then recurse down to the point where the sublist of L1 is empty []. Here, we start going back up, adding the elements of list L1 to the resultant list L3, at the front. Note that the second list L2 is kept untouched and is assigned to L3 at the base case of recursion.

11. Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.

```
bridges@bridges-CF-C2AHCCZC7: ~
GNU nano 4.8 TA_2_Prob_11.pl
concat([],L,L).
concat([X|L1],L2,[X|L3]):-concat(L1,L2,L3).

reverse([],[]).
reverse([X|L],R):-reverse(L,L1),concat(L1,[X],R).

```

```
bridges@bridges-CF-C2AHCCZC7: ~
bridges@bridges-CF-C2AHCCZC7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/home/bridges/TA_2_Prob_11.pl'].
true.

?- reverse([1,2,3,4,5],L).
L = [5, 4, 3, 2, 1].

?- 

```

## OBSERVATION/COMMENTS

1. Two cases emerge in concatenation of lists:

- case 1: list is empty.
- case 2: list is non-empty.

2. In case 1, we reverse of an empty list is the list itself.

3. In case 2, we separate the head of list L and then recurse down into the sublist of L. When the reverse is obtained from the recursive steps, the head that was extracted in the beginning is now appended to the list. The recursive steps themselves apply the same algorithm, till the base cases are encountered.

12. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.

```
bridges@bridges-CF-C2AHCCZ7: ~
GNU nano 4.8 TA_2_Prob_12.pl
concat([],L,L).
concat([X|L1],L2,[X|L3]):-concat(L1,L2,L3).

palindrome([]).
palindrome([_]).
palindrome(P):-concat([H|T],[H],P),palindrome(T).

```

```
bridges@bridges-CF-C2AHCCZ7: ~
bridges@bridges-CF-C2AHCCZ7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['/home/bridges/TA_2_Prob_12.pl'].
true.

?- palindrome([1,2,3,4,5]).
false.

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

?- 

```

## OBSERVATION/COMMENTS

1. Two cases emerge in concatenation of lists:

- case 1: list is empty.
- case 2: list has 1 element.
- case 3: list has more than 1 element.

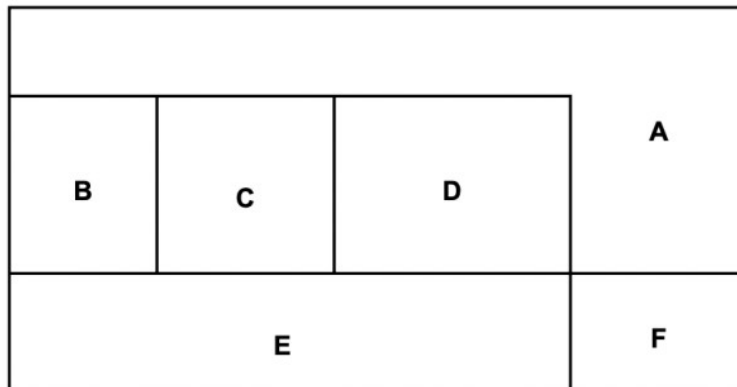
2. In cases 1 and 2, the list is a palindrome.

3. In case 3 the program *essentially* removes the end elements of the list in a recursive manner and checks if they are equal. Otherwise, the discrepancy is observed and *false* is the output.



### PART C : Exploratory Problem : Map Colorings

There is a famous problem in mathematics for coloring adjacent planar regions. Like cartographic maps, it is required that, whatever colors are used, no two adjacent regions may not have the same color. Two regions are considered adjacent provided they share some boundary line segment. Consider the following map.



Develop a Prolog program that can compute all possible colorings (Given colors to color with) are Red ,Blue, Green and Yellow. [ Hint : Covert it to graph first]

```
File Edit Selection View Go Run Terminal Help
TA_2_Prob_13.pl x
home > bridges > TA_2_Prob_13.pl
1 edge(a,b).
2 edge(a,c).
3 edge(a,d).
4 edge(a,f).
5
6 edge(b,a).
7 edge(b,c).
8 edge(b,e).
9
10 edge(c,a).
11 edge(c,b).
12 edge(c,d).
13 edge(c,e).
14
15 edge(d,a).
16 edge(d,c).
17 edge(d,e).
18
19 edge(e,b).
20 edge(e,c).
21 edge(e,d).
22 edge(e,f).
23
24 edge(f,a).
25 edge(f,e).
26
27 colour(red).
28 colour(green).
29 colour(blue).
30 colour(yellow).
```

```

31
32 has_colour(Node,Colour).
33
34 colorCode([],[]).
35 colorCode([Node|Nodes], [Code|Codes]) :-
36     colorCode(Nodes, Codes),
37     Code = has_colour(Node, Colour),
38     colour(Colour),
39     noconflict(Code, Codes).
40
41 noconflict(_,[]).
42 noconflict(Code1,[Code2|Codes]) :-
43     not(conflict(Code1,Code2)),
44     noconflict(Code1,Codes).
45
46 conflict(has_colour(A,Colour),has_colour(B,Colour)):-edge(A,B).

```

```

bridges@bridges-C2AHCCZC7:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

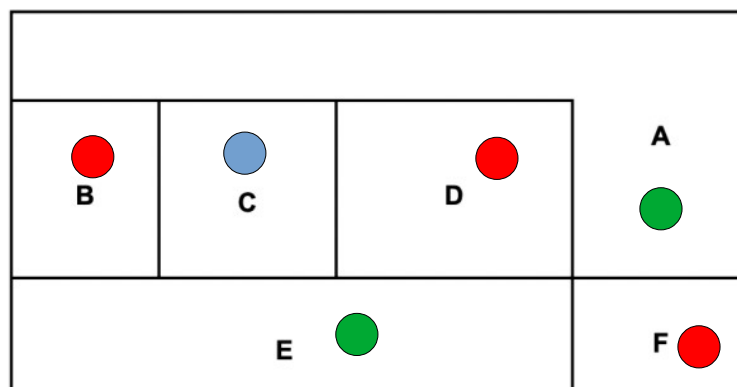
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).







?- ['/home/bridges/TA_2_Prob_13.pl'].
Warning: /home/bridges/TA_2_Prob_13.pl:32:
Warning: Singleton variables: [Node,Colour]
true.







?- colorCode([a,b,c,d,e,f],Colors).
Colors = [has_colour(a, green), has_colour(b, red), has_colour(c, blue), has_colour(d, red), has_colour(e, green), has_colour(f, red)] ;
Colors = [has_colour(a, yellow), has_colour(b, red), has_colour(c, blue), has_colour(d, red), has_colour(e, green), has_colour(f, red)] ;
Colors = [has_colour(a, green), has_colour(b, yellow), has_colour(c, blue), has_colour(d, red), has_colour(e, green), has_colour(f, red)] ;
Colors = [has_colour(a, green), has_colour(b, red), has_colour(c, yellow), has_colour(d, red), has_colour(e, green), has_colour(f, red)] .







```

There are more solutions for the graph coloring problem, only 4 are shown here:



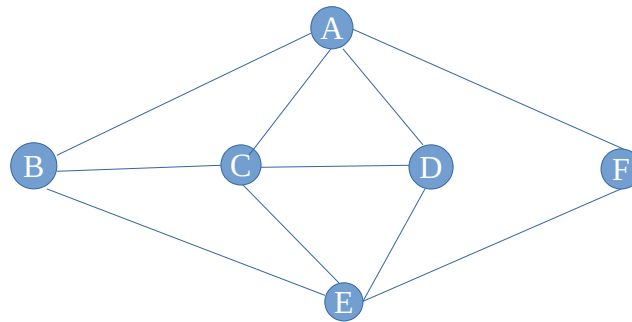
<div>B</div> <div></div>	<div>C</div> <div></div>	<div>D</div> <div></div>	<div>A</div> <div></div>
<div>E</div> <div></div>			<div>F</div> <div></div>

<div>B</div> <div></div>	<div>C</div> <div></div>	<div>D</div> <div></div>	<div>A</div> <div></div>
<div>E</div> <div></div>			<div>F</div> <div></div>

<div>B</div> <div></div>	<div>C</div> <div></div>	<div>D</div> <div></div>	<div>A</div> <div></div>
<div>E</div> <div></div>			<div>F</div> <div></div>

## OBSERVATION/COMMENTS

The map is first converted to a graph.



Where  $\text{edge}(X,Y)$  means regions X and Y share a common boundary.

1. The colorCode procedure first assigns a color to the node and then it checks for any conflicts.
2. The noconflict part checks whether there is a conflict in a particular code with the head of the list storing the codes for other nodes. It then recurses to the end of the list.
3. the conflict procedure will actually check the conflict case, where two adjacent nodes will have same colour.
4. Although the map was given in question, which was converted to graph, we can take the graph as an input by taking the ordered pairs of edges as input, and add extra code to determine edges and vertices.