



Sai
SAI RAM ENGINEERING COLLEGE

An Autonomous Institution | Affiliated to Anna University & Approved by AICTE, New Delhi
Accredited by NBA and NAAC "A+" | An ISO 9001:2015 Certified and MHRD NIRF ranked institution
Sai Leo Nagar, West Tambaram, Chennai - 600 044. www.sairam.edu.in

Founder Chairman : MJF. Ln. Leo Muthu



Name : _____

Register Number : _____

Laboratory Record

20CSPL601 – ARTIFICIAL INTELLIGENCE

LABORATORY

IV Year / VII Semester

ACADEMIC YEAR: 2024–25(ODD)

DEPARTMENT OF
ELECTRONICS & COMMUNICATION ENGINEERING



Sri SAI RAM ENGINEERING COLLEGE

An Autonomous Institution | Affiliated to Anna University & Approved by AICTE, New Delhi

Accredited by NBA and NAAC "A+" | An ISO 9001:2015 Certified and MHRD NIRF ranked institution

Sai Leo Nagar, West Tambaram, Chennai - 600 044. www.sairam.edu.in

Founder Chairman : MJF. Ln. Leo Muthu



Certificate

Register No. : _____

Certified that this is the Bonafide Record of work done by
Mr./Ms. _____
in the _____ Degree Course _____
in the _____
laboratory during the academic year _____

Station : Chennai - 600044

Date :

STAFF IN CHARGE

HEAD OF THE DEPARTMENT

Submitted for University Practical Examination held on _____ at
Sri Sai Ram Engineering College, Chennai – 600 044.

INTERNAL EXAMINER

EXTERNAL EXAMINER

S.NO	DATE	LIST OF EXPERIMENTS	PAGE NO	SIGNATURE
1		Study of Prolog	04	
2		Write Simple fact for the Statements using Prolog	06	
3		Write predicates one convert's centigrade temperature to Fahrenheit, other checks if a temperature is below freezing.	07	
4		Write a program to solve 4-Queen problem	09	
5		Write a program to solve 8-puzzle problem	11	
6		Write a program to solve any problem using Breadth First Search.	14	
7		Write a program to solve any problem using Depth First Search.	17	
8		Write a program to solve Travelling salesman Problem	19	
9		Write a prolog program to solve water jug problem	21	
10		Write a program to solve missionaries and cannibal problem	24	
11		Library Management System	27	
12		Simple ChatBot using Python	30	

Ex.no : 01 Date:	Study of Prolog
-----------------------------------	------------------------

Aim:

To study about prolog.

Introduction

Prolog stands for Programming in Logic - an idea that emerged in the early 1970's to use logic as programming language. The early developers of this idea included Robert Kowalski at Edinburgh (on the theoretical side), Marriten van Emden at Edinburgh (experimental demonstration) and Alian Colmerauer at Marseilles (implementation). David D.H. Warren's efficient implementation at Edinburgh in the mid -1970's greatly contributed to the popularity of PROLOG.

PROLOG is a programming language centered over a small set of basic mechanisms including pattern matching, tree based data structuring and automatic backtracking. This small set constitutes a surprisingly powerful and flexible programming framework. PROLOG is especially well suited for problems that involve objects in particular, structured objects and relations between them.

Prolog Clauses

- Any factual expression in prolog is called a clause.
- There are two types of factual expressions: facts and rules
- There are three categories of statements in prolog:

Facts: Those are true statements that form the basis for the knowledge base. Some facts about family relationships could be written as:

```
sister( sue,bill) parent( ann.sam) male(jo)
female( riya)
```

Rules: Similar to functions in procedural programming (C++, Java...) and has the form of if/then. To represent the general rule for grandfather, we write:

```
grand f.gher( X2) parent(X,Y) parent( Y,Z) male(X)
```

Queries: Questions that are passed to the interpreter to access the knowledge base and start the program. Given a database of facts and rules such as that above, we may make queries by typing afterba query a symbol'?' statements such as:

?-parent(X,sam) X=ann
?-grandfather(X,Y) X=john, Y=sam
Facts

Syntax rules:

1. The names of all relationships and objects must begin with a lower case letter.
For example: likes,john, raichel.
2. The relationship is written first, and the objects are written separated by commas, and the objects are enclosed by a pair of round brackets.
3. The character '.' must come at the end of each fact.

Prolog Program

Prolog is used for solving problems that involve objects and the relationships between objects. A program consists of a database containing one or more facts and zero or more rules. A fact is a relationship among a collection of objects. A fact is a one-line statement that ends with a full-stop.
parent (john, bart). parent (barbara, bart). male (john).

Meta Programming

A meta-program is a program that takes other programs as data. Interpreters and compilers are examples of meta programs. Meta-interpreter is a particular kind of meta-program: an interpreter for a language written in that language. So, a prolog interpreter is an interpreter for prolog, itself written in prolog. Due to its symbol-manipulation capabilities, prolog is a powerful language for meta-programming. Therefore, it is often used as an implementation language for other languages. Prolog is particularly suitable as a language for rapid prototyping where we are interested in implementing new ideas quickly. New ideas are rapidly implemented and experimented with.

Result:

Thus, Basics of Prolog is studied successfully.

Ex.no : 02 Date:	Write simple fact for the statements using prolog
-----------------------------------	--

Aim:

To write a program for simple fact statements using prolog.

- Ram likes mango.
- Seema is a girl.
- Bill likes Cindy.
- Rose is red.
- John owns gold.

Program:

Clauses

likes(ram,mango). girl(seema). red(rose). likes(bill ,cindy). owns(john ,gold)

Output Screenshot:

```
?-
% c:/users/administrator.dell-3/documents/prolog/sec21ec117 compiled 0.00 sec, -2 clauses
?- likes(ram,What).
What = mango.

?- liked(Who,cindy).
Who = bill.

?- red(What).
What = rose.

?- owns(Who,What).
Who = john,
What = gold.

?- |
```

Result:

Thus simple fact for the statements using the prolog program was executed successfully.

Ex.no : 03 Date:	Write predicates one convert's centigrade temperature to Fahrenheit, other checks if a temperature is below freezing.
-----------------------------------	--

Aim:

To write a prolog program to predict one convert's centigrade temperature to Fahrenheit and check if a temperature is below freezing or not.

Prolog programming: Centigrade and Fahrenheit Temperatures

The centigrade scale, which is also called the Celsius scale, was developed by Swedish astronomer Andres Celsius. In the centigrade scale, water freezes at 0 degrees and boils at 100 degrees. The centigrade to Fahrenheit conversion formula is:

Fahrenheit and centigrade are two temperature scales in use today. The Fahrenheit scale was developed by the German physicist Daniel Gabriel Fahrenheit. In the Fahrenheit scale, water freezes at 32 degrees and boils at 212 degrees.

Algorithm:

Step1: Start the program

Step2: Read the input of temperature in Celsius

Step3: $F = (9 \times C) / 5 + 32$

Step4: Print temperature in Fahrenheit is F

Step5: Stop the program

Program:

Production rules: Arithmetic:

c_to_f f is $c * 9 / 5$

+32 freezing $f \leq 32$

Rules:

$c_to_f(C,F)$:-

F is $C * 9 / 5 + 32$.

freezing(F) :- $F \leq 32$.

Output Screenshot:

```
?-  
% c:/users/administrator.dell-3/documents/prolog/sec21ec1172 compiled 0.00 sec, 0 clauses  
?- c_to_f(100,X).  
X = 212.  
  
?- freezing(15).  
true.  
  
?-
```

Result:

Thus the conversion of centigrade temperature to Fahrenheit program was executed successfully.

Ex.no :04 Date:	Write a program to solve 4-Queen problem
----------------------------------	---

Aim:

To write a program to solve 4 – queen problems using prolog.

Algorithm:

Step1: Start the program from leftmost column

Step2: If all queens are placed return true

Step3: Try all rows in the current column. Do the following for every tried row.

- a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
- b) If placing the queen in [row, column] leads to a solution then return true.
- c) If placing queen doesn't lead to a solution then unmark this [row, column] and go to step (a) to try other rows.

Step4: If all rows have been tried and nothing worked, return false.

Step5: Stop the program.

The 4-Queens Problem consists in placing four queens on a 4 x 4 chessboard so that no two queens can capture each other. That is, no two queens are allowed to be placed on the same row, the same column or the same diagonal. The following figure illustrates a solution to the 4 - Queens Problem: none of the 4 queens can capture each other.

Program:

```
perm([X|Y],Z) :- perm(Y,W), takeout(X,Z,W). perm([],[]).
takeout(X,[X|R],R). takeout(X,[F|R],[F|S]) :- takeout(X,R,S).
solve(P) :- perm([1,2,3,4],P),
combine([1,2,3,4],P,S,D),
all_diff(S), all_diff(D).
combine([X1|X],[Y1|Y],[S1|S],[D1|D]) :- S1 is X1
+Y1,D1 is X1 - Y1,
combine(X,Y,S,D). combine([],[],[],[]).
all_diff([X|Y]) :- \+member(X,Y), all_diff(Y). all_diff([X]).
```

	1	2	3	4
1			Q	
2	Q			
3				Q
4		Q		

Output Screenshot:

```

File Edit Settings Run Debug Help
% c:/users/administrator.dell-3/documents/prolog/sec21ec1173 compiled 0.00 sec, 0 clauses
creep
?- solve(P).
P = [3, 1, 4, 2] .

?- setof(P,solve(P),Set),length(Set,L).
Set = [[2, 4, 1, 3], [3, 1, 4, 2]],
L = 2.

?-

```

Result:

Thus the program for the 4 queen problem is executed and the output is obtained successfully.

Ex.no : 05 Date:	Write a program to solve 8-puzzle problem
-----------------------------------	--

Aim:

To write a program to solve 8 – puzzle problems using prolog.

Algorithm

Step 1: Start the program and represent the board position as 8*8 vectors, i.e., [1,2,3,4,5,6,7,8]. Store the set of queens in the list 'Queens'.

Step 2: Calculate the permutation of the above eight numbers stored in set P.

Step 3: Let the position where the first queen to be placed be(1,Y), for second be (2,Y1),and so on and store the position in S.

Step 4: Check for the safety of the queens through the predicate, 'noattack()'.

Step 5: Calculate Y1-y and Y-Y1. If both are not equal to Xdist, which is the X- distance between the first queen and others, then goto step 6 else goto step 7.

Step 6: Increment Xdist by 1.

Step 7: Repeat above for the rest of the queens, until the end of the list is reached.

Step 8: Print S as answer.

Step 9: Stop the program.

Program:

test(Plan):-

write('Initial state:'),nl,

Init= [at(empty,1), at(tile8,2), at(tile1,3), at(tile5,4), at(tile3,5), at(tile2,6), at(tile7,7), at(tile4,8), at(tile6,9)],

write_sol(Init),

Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(tile5,5), at(tile6,6), at(tile7,7), at(tile8,8), at(empty,9)],

nl,write('Goal state:'),nl,

write(Goal),nl,nl,

solve(Init,Goal,Plan).

solve(State, Goal, Plan):-

solve(State, Goal, [], Plan).

is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

solve(State, Goal, Plan, Plan):-

is_subset(Goal, State), nl,

write_sol(Plan).

solve(State, Goal, Sofar, Plan):-

```

act(Action, Preconditions, Delete, Add),
is_subset(Preconditions, State),
\+ member(Action, Sofar), delete_list(Delete,
State, Remainder), append(Add, Remainder,
NewState), solve(NewState, Goal,
[Action|Sofar], Plan). act(move(X,Y,Z),
[at(X,Y), at(empty,Z), is_movable(Y,Z)],
[at(X,Y), at(empty,Z)],
[at(X,Z), at(empty,Y)]).
is_subset([H|T], Set):-
member(H, Set),
is_subset(T, Set).
is_subset([], _).
delete_list([H|T], Curstate, Newstate):-
remove(H, Curstate, Remainder),
delete_list(T, Remainder, Newstate).
delete_list([], Curstate, Curstate).
remove(X, [X|T], T).
remove(X, [H|T], [H|R]):-
remove(X, T, R).
write_sol([]).
write_sol([H|T]):-
write_sol(T), write(H),
nl.
append([H|T], L1, [H|L2]):-
append(T, L1, L2).
append([], L, L).
member(X, [X|_]).
member(X, [_|T]):-
member(X, T).

```

Initial State			Goal State		
	8	1	1	2	3
5	3	2	4	5	6
7	4	6	7	8	

Output Screenshot:

```
?-
% c:/users/administrator.dsp-43/documents/prolog/puzzle8 compiled 0.00 sec, 0 clauses
?-
| test(Plan).
Initial state:
at(tile6,9)
at(tile4,8)
at(tile7,7)
at(tile2,6)
at(tile3,5)
at(tile5,4)
at(tile1,3)
at(tile8,2)
at(empty,1)

Goal state:
[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(tile5,5),at(tile6,6),at(tile7,7),at(tile8,8),at(empty,9)]

false.
```

Result:

Thus the program for the 8 puzzle problem using prolog is executed and the output is obtained successfully.

Ex.no : 06 Date:	Write a program to solve any problem using Breadth First Search
-----------------------------------	--

Aim:

To write a program to solve any problem using Breadth First Search using prolog.

Algorithm

Step 1: Start the program

Step 2: Enter the node to be found

Step 3: If the initial state is a goal state, quit and return success

Step 4: Otherwise, do the following until success or failure is signaled.

- a. Generate a successor, E, of the initial state. If there are no more successors,

Signal Failure.

- b. Call Breadth-First Search with E as the initial state.

- c. If success is returned, signal success. Otherwise continue in this loop.

Step 5: Print the output as the path traversed

Step 6: Stop the program.

Program:

s(a,b).

s(a,c).

s(b,d).

s(b,e).

s(c,f).

s(c,g).

s(d,h).

s(e,i).

s(e,j).

s(f,k).

goal(f).

goal(j).

solve(Start,Solution):-

bfs([[Start]],Solution).

bfs([[Node|Path]|_],[Node|Path]):-

goal(Node).

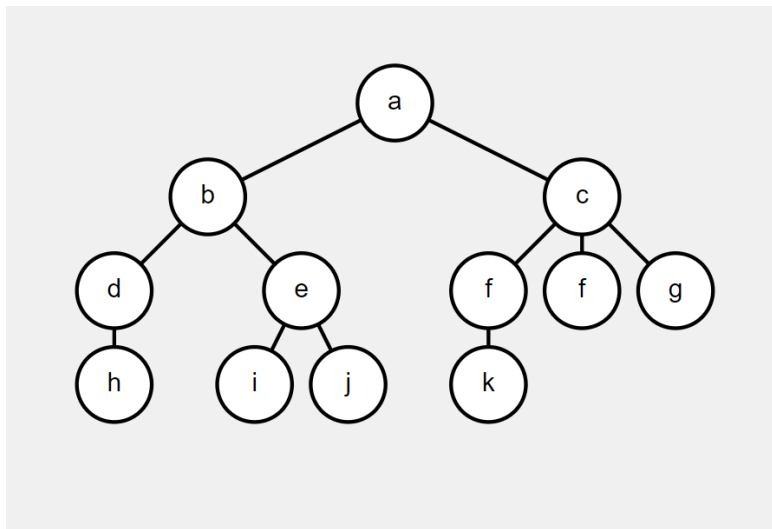
bfs([Path|Paths],Solution):-

```

extend(Path,NewPaths),
write(NewPaths),
nl,
conc(Paths,NewPaths,Paths1),bfs(P
aths1,Solution).
extend([Node|Path],NewPaths):-
bagof([NewNode,Node|Path],(s(Node,NewNode),not(member(NewNode,[Node|Path]))),NewPat
hs),!.
extend(_,[]).
conc([],L,L).
conc([X|L1],L2,[X|L3]):-nl,write('conc'),write(X),write('  '),write(L1),write(L2),conc(L1,L2,L3).

```

Tree Diagram :



Output Screenshot:

```
% c:/users/administrator.dell-3/documents/prolog/sec21ec1174 compiled 0.00 sec, 0 clauses
?- solve(a,S).
[[b,a],[c,a]]
[[d,b,a],[e,b,a]]

conc[c,a][][][[d,b,a],[e,b,a]][][[f,c,a],[f,c,a],[g,c,a]]

conc[d,b,a][][][[e,b,a]][][[f,c,a],[f,c,a],[g,c,a]]
conc[e,b,a][][][[f,c,a],[f,c,a],[g,c,a]][][[h,d,b,a]]

conc[e,b,a][][][[f,c,a],[f,c,a],[g,c,a]][][[h,d,b,a]]
conc[f,c,a][][][[f,c,a],[g,c,a]][][[h,d,b,a]]
conc[f,c,a][][][[g,c,a]][][[h,d,b,a]]
conc[g,c,a][][][[h,d,b,a]][][[i,e,b,a],[j,e,b,a]]

conc[f,c,a][][][[f,c,a],[g,c,a],[h,d,b,a]][][[i,e,b,a],[j,e,b,a]]
conc[f,c,a][][][[g,c,a],[h,d,b,a]][][[i,e,b,a],[j,e,b,a]]
conc[g,c,a][][][[h,d,b,a]][][[i,e,b,a],[j,e,b,a]]
conc[h,d,b,a][][][[i,e,b,a],[j,e,b,a]]
S = [f, c, a] |
```

Result:

Thus the program for breadth-first search using prolog is executed and the output is obtained successfully.

Ex.no : 07 Date:	Write a program to solve any problem using Depth First Search
-----------------------------------	--

Aim:

To write a program to solve any problem using Depth First Search using prolog.

Algorithm:

Step 1: Start the program

Step 2: Enter the node to be found

Step 3: If the initial state is a goal state, quit and return success

Step 4: Otherwise, do the following until success or failure is signaled.

a. Generate a successor, E, of the initial state. If there are no more successors,
Signal Failure

b. Call Depth -First Search with E as the initialstate.

c. If success is returned, signal success. Otherwise continue in this loop.

Step 5: Print the output as the path traversed

Step 6: Stop the program.

Program:

```

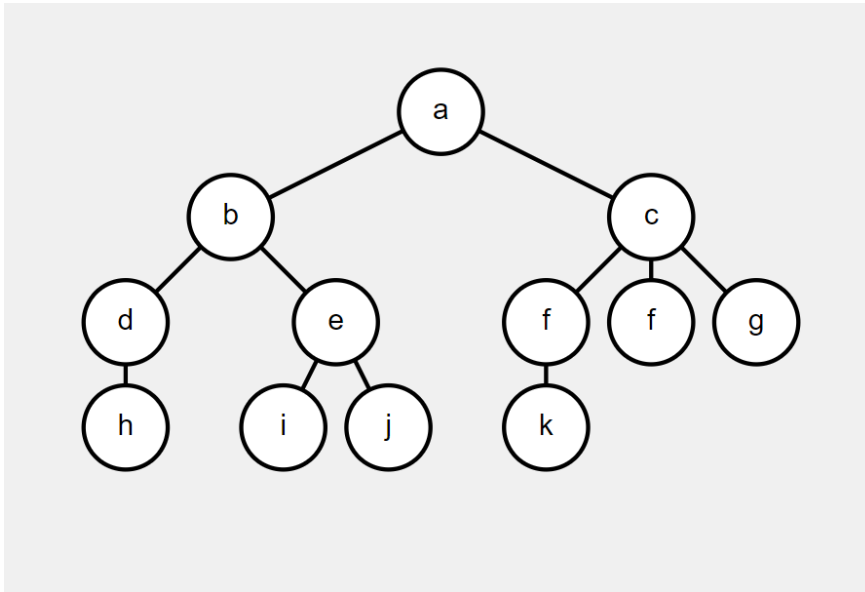
s(a,b).
s(a,c).
s(b,d).
s(b,e).
s(c,f).
s(c,g).
s(d,h).
s(e,i).
s(e,j).
s(f,k).
goal(f).
goal(j).
mem(X,[X|_]).
mem(X,[_|Tail]):-mem(X,Tail).
solve(Node,Solution):-
dfs([],Node,Solution).
dfs(Path,Node,[Node|Path]):-

```

```

goal(Node).
dfs(Path,Node,Sol):-
s(Node,Node1),
not(mem(Node1,Path)),
dfs([Node|Path],Node1,Sol).

```



Output Screenshot:

```
% c:/users/administrator.dell-3/documents/prolog/sec21ec1175 compiled 0.00 sec, 0 clauses
```

```
Unknown action: s (h for help)
```

```
Action?
```

```
Unknown action: o (h for help)
```

```
Action?
```

```
Unknown action: l (h for help)
```

```
Action?
```

```
Unknown action: v (h for help)
```

```
Action?
```

```
Unknown action: e (h for help)
```

```
Action? .
```

```
?- solve(a,S).
```

```
S = [h, d, b, a] ;
```

```
S = [g, c, a]
```

Result:

Thus the program for depth-first search using prolog is executed and the output is obtained successfully.

Ex.no : 8 Date:	Write a program to solve Travelling salesman Problem
----------------------------------	---

Aim:

To write a program to solve travelling salesman problem using prolog.

Algorithm

Step 1: Start the program

Step 2: Consider city 1 as the starting and ending point.

Step 3: Generate all $(n-1)!$ Permutations of cities.

Step 4: Calculate cost of every permutation and keep track of minimum cost permutation.

Step 5: Return the permutation with minimum cost.

Step 6: Stop the program.

Program:

```

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). get_road(Start, End, Visited,
Result) :-get_road(Start, End, [Start], 0, Visited, Result).
get_road(Start, End, Waypoints, DistanceAcc, Visited, TotalDistance) :- road(Start, End, Distance),
reverse([End|Waypoints], Visited), TotalDistance is DistanceAcc + Distance.
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).
```

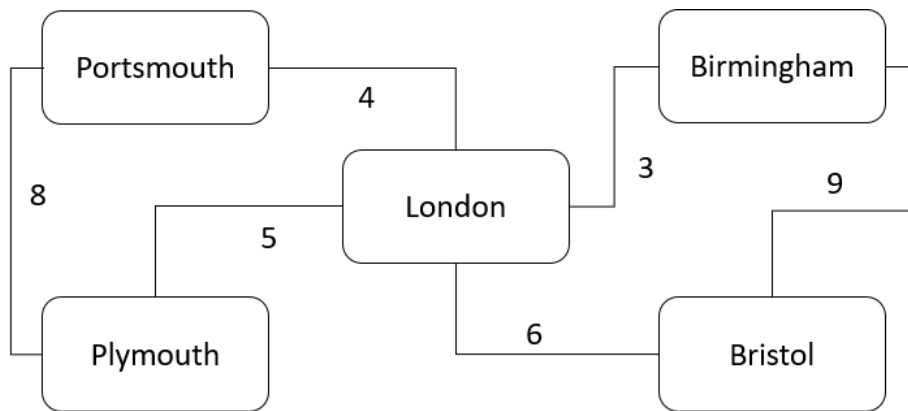
Output Screenshot:

```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.11)
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).

?-
% c:/users/administrator.dsp-43/documents/prolog/solvetravelling salesman prolog
?- get_road(portsmouth, plymouth, Visited, Distance).
Visited = [portsmouth, plymouth].
Distance = 8 .

?- Visited = [portsmouth,plymouth] ?
|
|
```



Result:

Thus the program for travelling salesman using prolog is executed and the output is obtained successfully.

Ex.no :09 Date:	Write a prolog program to solve water jug problem
----------------------------------	--

Aim:

To write a program to solve water jug problem using prolog.

Algorithm:

Step 1: Start the program

Step 2: Fill any of the jugs fully with water.

Step 3: Empty any of the jugs.

Step 4: Pour water from one jug into another till the other jug is completely full or the first jug itself is empty.

Step 5: Stop the program.

Program:

```

fill(x,y).
fill(2,0):-
nl,
write('Goal State is Reached    ').
fill(X,Y):-
X=0,
Y=<1,
nl,
write('Fill the 4-Gallon  Jug:(4,)',write(Y),write(')'),fill(4,Y).
fill(X,Y):-
Y=0,
X>=3,
nl,
write('Fill the 3-Gallon Jug:('),
write(X),
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 until is full:(4,)', write(Y1),
write(')'),
fill(4,Y1).
fill(X,Y):-
X+Y>=3,
X=4,
Y=<1,
X1 is X-(3-Y),
nl,
write('Pour water from 4-Gallon Jug to 3-Gallon until 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 the water from 3-Gallon Jug to 4-Gallon:('),
write(X1),
write(',0)'),
fill(X1,0).
fill(X,Y):-
X+Y<3,
Y=0,
Y1 is X+Y,
nl,
write('Pour all the water from 4-Gallon Jug too 3-Gallon:(0,)',
write(Y1),
write(')'),
fill(0,Y).
fill(X,Y):-
Y>=2,
X=4,
nl,
write('Empty the 4-Gallon Jug on Ground:(0,)',
write(Y),
write(')'),

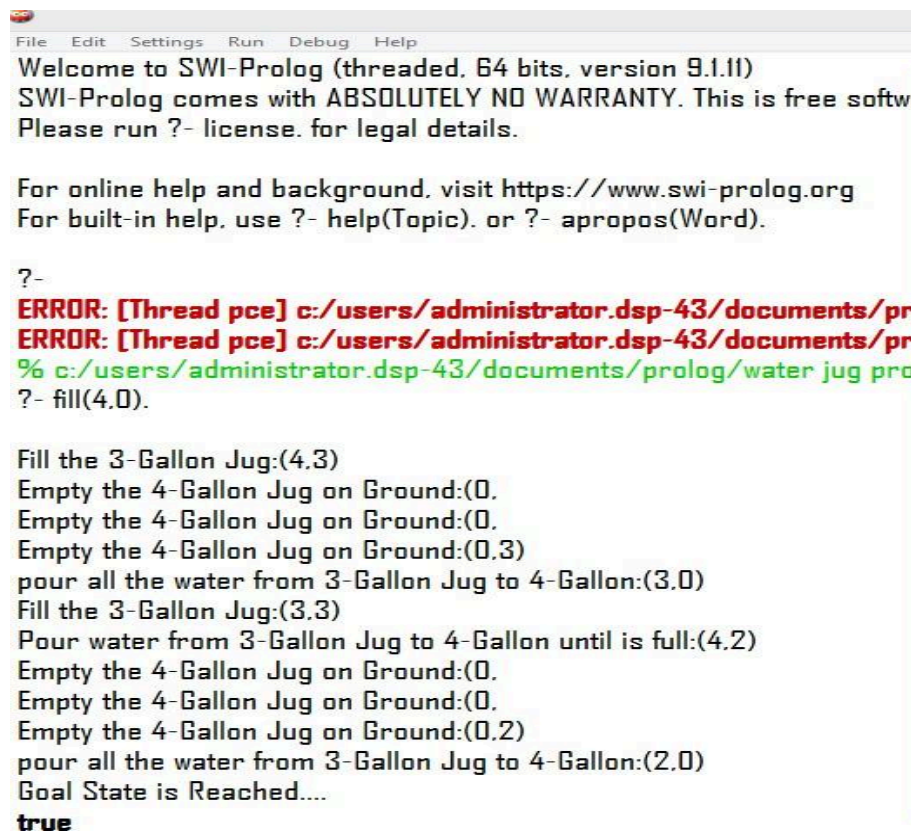
```

```

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('4L Jug Overflowed.').nl.
fill(X,Y):- X<4, Y>3,
write('3L Jug Overflowed.').nl.
fill(X,Y):-X>4,Y>3,
write('4L3L Jug Overflowed.').nl.

```

Output Screenshot:



```

File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.11)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free softw
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).

?-
ERROR: [Thread pce] c:/users/administrator.dsp-43/documents/pr
ERROR: [Thread pce] c:/users/administrator.dsp-43/documents/pr
% c:/users/administrator.dsp-43/documents/prolog/water jug pro
?- fill(4,0).

Fill the 3-Gallon Jug:(4,3)
Empty the 4-Gallon Jug on Ground:(0.
Empty the 4-Gallon Jug on Ground:(0.
Empty the 4-Gallon Jug on Ground:(0,3)
pour all the water from 3-Gallon Jug to 4-Gallon:(3,0)
Fill the 3-Gallon Jug:(3,3)
Pour water from 3-Gallon Jug to 4-Gallon until is full:(4,2)
Empty the 4-Gallon Jug on Ground:(0.
Empty the 4-Gallon Jug on Ground:(0.
Empty the 4-Gallon Jug on Ground:(0,2)
pour all the water from 3-Gallon Jug to 4-Gallon:(2,0)
Goal State is Reached....
true

```

Result:

Thus the program for the water jug problem using prolog is executed and the output is obtained successfully.

Ex.no : 10 Date:	Write a program to solve missionaries and cannibal problem
-----------------------------------	---

Aim:

To write a prolog program to solve Missionaries and Cannibal Problem.

Procedure:

The Missionaries and Cannibals problem is a classic problem in Artificial Intelligence that involves three missionaries and three cannibals on one side of a river, along with a boat that can carry at most two people at a time. The goal is to move all the people to the other side of the river without ever leaving more missionaries than cannibals on either side, or else the cannibals will eat the missionaries.

Program:

```
% Represent a state as [CL,ML,B,CR,MR]
start([3,3,left,0,0]).
goal([0,0,right,3,3]).
legal(CL,ML,CR,MR) :-
% is this state a legal one? ML>=0,
CL>=0, MR>=0, CR>=0,
(ML>=CL ; ML=0),
(MR>=CR ; MR=0).
% Possible moves:
move([CL,ML,left,CR,MR],[CL,ML2,right,CR,MR2]):-
% Two missionaries cross left to right.
MR2 is MR+2,
ML2 is ML-2,
legal(CL,ML2,CR,MR2).
move([CL,ML,left,CR,MR],[CL2,ML,right,CR2,MR]):-
% Two cannibals cross left to right.
CR2 is CR+2,
CL2 is CL-2,
legal(CL2,ML,CR2,MR).
move([CL,ML,left,CR,MR],[CL2,ML2,right,CR2,MR2]):-
% One missionary and one cannibal cross left to right.
CR2 is CR+1,
CL2 is CL-1,
```



```

MR2 is MR+1,
ML2 is ML-1,
legal(CL2,ML2
,CR2,MR2).
move([CL,ML,left,CR,MR],[CL,ML2,right,CR,MR2]):-
% One missionary crosses left to right.
MR2 is MR+1,
ML2 is ML-1,
legal(CL,ML2,CR,MR2).
move([CL,ML,left,CR,MR],[CL2,ML,right,CR2,MR]):-
% One cannibal crosses left to right.
CR2 is CR+1,
CL2 is CL-1,
legal(CL2,ML,CR2,MR).
move([CL,ML,right,CR,MR],[CL,ML2,left,CR,MR2]):-
% Two missionaries cross right to left.
MR2 is MR-2,
ML2 is ML+2,
legal(CL,ML2,CR,MR2).
move([CL,ML,right,CR,MR],[CL2,ML,left,CR2,MR]):-
% Two cannibals cross right to left.
CR2 is CR-2,
CL2 is CL+2,
legal(CL2,ML,CR2,MR).
move([CL,ML,right,CR,MR],[CL2,ML2,left,CR2,MR2]):-
% One missionary and one cannibal cross right to left.
CR2 is CR-1,
CL2 is CL+1,
MR2 is MR-1,
ML2 is ML+1,
legal(CL2,ML2,CR2,MR2).
move([CL,ML,right,CR,MR],[CL,ML2,left,CR,MR2]):-
% One missionary crosses right to left.
MR2 is MR-1,
ML2 is ML+1,
legal(CL,ML2,CR,MR2).
move([CL,ML,right,CR,MR],[CL2,ML,left,CR2,MR]):-
% One cannibal crosses right to left.
CR2 is CR-1,
CL2 is CL+1,

```

```

legal(CL2,ML,CR2,MR).
% Recursive call to solve the problem
path([CL1,ML1,B1,CR1,MR1],[CL2,ML2,B2,CR2,MR2],Explored,MovesList) :-
move([CL1,ML1,B1,CR1,MR1],[CL3,ML3,B3,CR3,MR3]),
not(member([CL3,ML3,B3,CR3,MR3],Explored)),
path([CL3,ML3,B3,CR3,MR3],[CL2,ML2,B2,CR2,MR2],[[CL3,ML3,B3,CR3,MR3]|Explored],
[ [[CL3,ML3,B3,CR3,MR3],[CL1,ML1,B1,CR1,MR1]] | MovesList ] ).
% Solution found
path([CL,ML,B,CR,MR],[CL,ML,B,CR,MR],_,MovesList):-
output(MovesList).
% Printing
output([]) :- nl.
output([[A,B]|MovesList]) :-
output(MovesList),
write(B), write(' -> '), write(A), nl.
% Find the solution for the missionaries and cannibals problem find:-
path([3,3,left,0,0],[0,0,right,3,3],[[3,3,left,0,0]],_).

```

Output Screenshot:

```

% c:/users/administrator.dell-3/documents/prolog/exp9man compiled 0.00 sec, 0 clauses
?- find.

[3,3,left,0,0] -> [1,3,right,2,0]
[1,3,right,2,0] -> [2,3,left,1,0]
[2,3,left,1,0] -> [0,3,right,3,0]
[0,3,right,3,0] -> [1,3,left,2,0]
[1,3,left,2,0] -> [1,1,right,2,2]
[1,1,right,2,2] -> [2,2,left,1,1]
[2,2,left,1,1] -> [2,0,right,1,3]
[2,0,right,1,3] -> [3,0,left,0,3]
[3,0,left,0,3] -> [1,0,right,2,3]
[1,0,right,2,3] -> [1,1,left,2,2]
[1,1,left,2,2] -> [0,0,right,3,3]
true .

```

Result:

Thus the program to solve missionaries and cannibal problem using prolog is executed and the output is obtained successfully.

Ex.no : 11 Date:	Library Management System
-----------------------------------	----------------------------------

Aim:

To write a Prolog program to simulate Library Management System.

Algorithm:

Step 1: Define Book and Student Information. Set up predicates to store book details (title, author, publication year, and stock) and student names.

Step 2: Record Book Transactions. Create a predicate to record when a student takes a book, including the book title, students name, and dates of borrowing and returning.

Step 3: Define Rules for Queries. Create rules to find the author of a book, check the number of copies in stock, and calculate available copies.

Step 4: User Queries. Allow users to ask questions based on the defined rules, such as querying the author of a book or checking the available copies.

Step 5: Match Queries with Book Information. When a query is made, match it with stored book details to find relevant information.

Step 6: Match Queries with Student Information. If the query involves student data (e.g., books taken by a specific student), match it with stored student records.

Step 7: Apply Rules to Calculate Available Copies. For queries about available copies, apply the rule to subtract taken copies from the total stock, providing the result.

Step 8: Return Query Results. Return the results of the queries to the user, providing answers to their questions.

Step 9: Handle Multiple Queries. Allow for multiple queries to be executed in sequence, processing each one and returning the appropriate responses.

Step 10: End Program. End the program or continue to accept user queries, providing information as long as the program is running.

Program:

% Facts

book(The Hobbit, J.R.R. Tolkien, 1937, 2). % 2 copies in stock

book(1984, George Orwell, 1949, 3). % 3 copies in stock

book(To Kill a Mockingbird, Harper Lee, 1960, 5). % 5 copies in stock

% Additional 10 books

% (Books omitted for brevity)

% Additional 10 books

```

book(The Catcher in the Rye, J.D. Salinger, 1951, 1).
book(Pride and Prejudice, Jane Austen, 1813, 4). book(The Great Gatsby, F. Scott Fitzgerald, 1925, 2).
book(Moby-Dick, Herman Melville, 1851, 3). book(The Lord of the Rings, J.R.R. Tolkien, 1954, 1).
book(One Hundred Years of Solitude, Gabriel Garcia Marquez, 1967, 2). book(Brave
New World, Aldous Huxley, 1932, 1).
book(Crime and Punishment, Fyodor Dostoevsky, 1866, 2).
book(The Odyssey, Homer, -800, 1).
book(Frankenstein, Mary Shelley, 1818, 1).
% Student records
student(Alice).
student(Bob).
student(Charlie).
student(David).
student(Eva).
student(Frank).
student(Grace).
student(Henry).
student(Ivy).
student(Jack).
% Taken books
taken(The Hobbit, Alice, 2023-10-25, 2023-11-10).
taken(1984, Bob, 2023-10-20, 2023-11-05).
taken(To Kill a Mockingbird, Charlie, 2023-10-22, 2023-11-08).
taken(Pride and Prejudice, Eva, 2023-10-21, 2023-11-07).
taken(The Great Gatsby, David, 2023-10-23, 2023-11-09).
% Rules
author(Author, Title) :- book(Title, Author, _, _).
in_stock(Title, InStock) :- book(Title, _, _, InStock).
% Calculate available books
available(Title, Available) :- book(Title, _, _, Total),
findall(Student, taken(Title, Student, _, _), Taken),
length(Taken, TakenCount),
Available is Total - TakenCount.
% Queries
% Example: Who is the author of The Hobbit?
% Query: author(Author, The Hobbit).
% Example: How many copies of The Hobbit are available?
% Query: available(The Hobbit, Available).
%Query: in_stock(The Hobbit, InStock).
%Query: taken(1984, Student, TakenDate, ReturnDate).

```

Output Screenshot:

```
% e:/laboratory/sem vii lab/artificial intelligence laboratory/prolog/exp1 lman compiled 0.00 sec, 24 clauses
?- author(Author, the_hobbit).
Author = tolkien.

?- available(the_hobbit, Available).
Available = 1.

?- in_stock(1984, InStock).
InStock = 3.

?- taken(1984, Student, TakenDate, ReturnDate).
Student = bob,
TakenDate = date(2023, 10, 20),
ReturnDate = date(2023, 11, 5).

?- |
```

Result:

Thus, the Library Management system is simulated using Prolog.

Ex.no : 12

Date:

Simple ChatBot using Python

Aim:

To develop a simple ChatBot using Python.

Algorithm:

Step 1: Import necessary modules from the `chatterbot` library: `ChatBot` and `ListTrainer`.

Step 2: Create a `ChatBot` instance named "chatbot" with specific configurations: `read_only=False` and `logic_adapters=["chatterbot.logic.BestMatch"]`.

Step 3: Define conversational pairs for training in a list (`list_to_train`).

Step 4: Create a `ListTrainer` instance and associate it with the previously created `chatbot`.

Step 5: Train the chatbot using `list_trainer.train(list_to_train)` with the prepared conversational data.

Step 6: Start an infinite loop (`while True`) for chatting.

Step 7: Accept user input using `input("User: ")`.

Step 8: Check for the "stop" condition; if input is "stop", break the loop.

Step 9: Get the bot's response to the user input using `bot.get_response(user_text)`.

Step 10: Display the bot's response for the user input using `print()`.

Program:

```
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
bot = ChatBot("chatbot", read_only=False, logic_adapters=["chatterbot.logic.BestMatch"])
list_to_train = [
    "Hi",
    "Hi there",
    "What's your name?",
    "I'm a chatbot", "How
old are you?", "I'm
ageless!",
    "Why are you so mad?",
    "I'm not!",
    "Do you have iPhone?",
    "I've everything!",
    "What's your favorite food?", "I
```

```

don't eat",

"What's your job?",
"I'm here to answer your questions",
]
list_trainer = ListTrainer(bot)
list_trainer.train(list_to_train)
while True:
    user_text = input("User: ")
    if(user_text == "stop"):
        break
    print("Bot: " + str(bot.get_response(user_text)))

```

Output:

```

List Trainer: [#####] 100%
User: Hi
Bot: Hi there
User: What's your name?
Bot: I'm a chatbot
User: How old are you?
Bot: I'm ageless!
User: What's your job?
Bot: I'm here to answer your questions
User: What's your favorite food?
Bot: I don't eat
User: Do you have iPhone?
Bot: I've everything!
User: stop

Process finished with exit code 0
|

```

Result:

Thus, the simple ChatBot using Python has been developed successfully.