



Sri

SAI RAM ENGINEERING COLLEGE

An Autonomous Institution | Affiliated to Anna University & Approved by AICTE, New Delhi
Accredited by NBA and NAAC "A+" | BIS/EOMS ISO 21001 : 2018 and BVQI 9001 : 2015 Certified and NIRF ranked institution
Sai Leo Nagar, West Tambaram, Chennai - 600 044. www.sairam.edu.in



PROSPERITY THROUGH TECHNOLOGY

LAB MANUAL

20CSPL601-ARTIFICIAL INTELLIGENCE LABORATORY

IV YEAR VII SEMESTER

Academic Year: 2025-2026

**Department of
Electronics and Communication Engineering**

PREFACE

“THE TRUE METHOD OF KNOWLEDGE IS EXPERIMENT”

The true transmission of knowledge really occurs when the student is open to and engaged in the information. With this in mind, this manual is compiled as a preparatory note for the Artificial Intelligence laboratory experiments. Sufficient details have been included to impart self-learning.

Artificial intelligence (AI) as one of the most important topics of the digital future, is currently experiencing a huge upward trend in all fields of science, economy, technology, health care and entertainment. It aims in Identifying innovative research directions in Artificial Intelligence, Deep Learning, Machine Learning, and Big Data analytics.

This manual is intended for the VII semester students of ECE . Each experiment is provided with introductory information and procedure to perform the experiment.

Head of the Department

Principal

INSTITUTION VISION

To emerge as a "Centre of excellence" offering Technical Education and Research opportunities of very high standards to students, develop the total personality of the individual and instill high levels of discipline and strive to set global standards, making our students technologically superior and ethically stronger, who in turn shall contribute to the advancement of society and humankind.

INSTITUTION MISSION

We dedicate and commit ourselves to achieve, sustain and foster unmatched excellence in Technical Education. To this end, we will pursue continuous development of infra-structure and enhance state-of-art equipment to provide our students a technologically up-to date and intellectually inspiring environment of learning, research, creativity, innovation and professional activity and inculcate in them ethical and moral values.

INSTITUTION POLICY

We at Sri Sai Ram Engineering College are committed to build a better Nation through Quality Education with team spirit. Our students are enabled to excel in all values of Life and become Good Citizens. We continually improve the System, Infrastructure and Service to satisfy the Students, Parents, Industry and Society.

DEPARTMENT VISION

To emerge as a "centre of excellence" in the field of Electronics and Communication Engineering and to mould our students to become technically and ethically strong to meet the global challenges. The Students in turn contribute to the advancement and welfare of the society.

DEPARTMENT MISSION

M1: To achieve, sustain and foster excellence in the field of Electronics and Communication Engineering.

M2: To adopt proper pedagogical methods to maximize the knowledge transfer.

M3: To enhance the understanding of theoretical concepts through professional society activities

M4: To improve the infrastructure and provide conducive environment of learning and research following ethical and moral values

Program Educational Objectives (PEOs)

To prepare the graduates to:

1. Acquire strong foundation in Engineering, Science and Technology for a successful career in Electronics and Communication Engineering.
2. Apply their knowledge and skills acquired to solve the issues in real world Electronics and Communication sectors and to develop feasible and viable systems.
3. Be receptive to new technologies and attain professional competence through professional society activities.
4. Participate in lifelong learning, higher education efforts to emerge as expert researchers and technologists.
5. Practice the profession with ethics, integrity, leadership and social responsibilities

PROGRAM OUTCOMES (POS)

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to

comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Electronics and Communication Engineering graduates will be able to:

1. Design, implement and test Electronics and Communication systems using analytical knowledge and applying modern hardware and software tools
2. Develop their skills to solve problems and assess social, environmental issues with ethics and manage different projects in multidisciplinary areas.

OBJECTIVES:

- To learn Prolog Programming
- To implement different search strategies and solve game problems in AI using Prolog
- To implement Natural Language Processing in AI applications

OUTCOMES:

On completion of this laboratory course, the student should be able to

1. Apply informed and uninformed search strategies to solve AI problems.(K3)
2. Select State Space Searching method to solve AI problems.(K3)
3. Demonstrate an application using Natural Language Processing. (K3)

DATE:	STUDY OF PROLOG
EXP NO: 1	

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), Marten van Emden at Edinburgh (experimental demonstration) and Alan 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

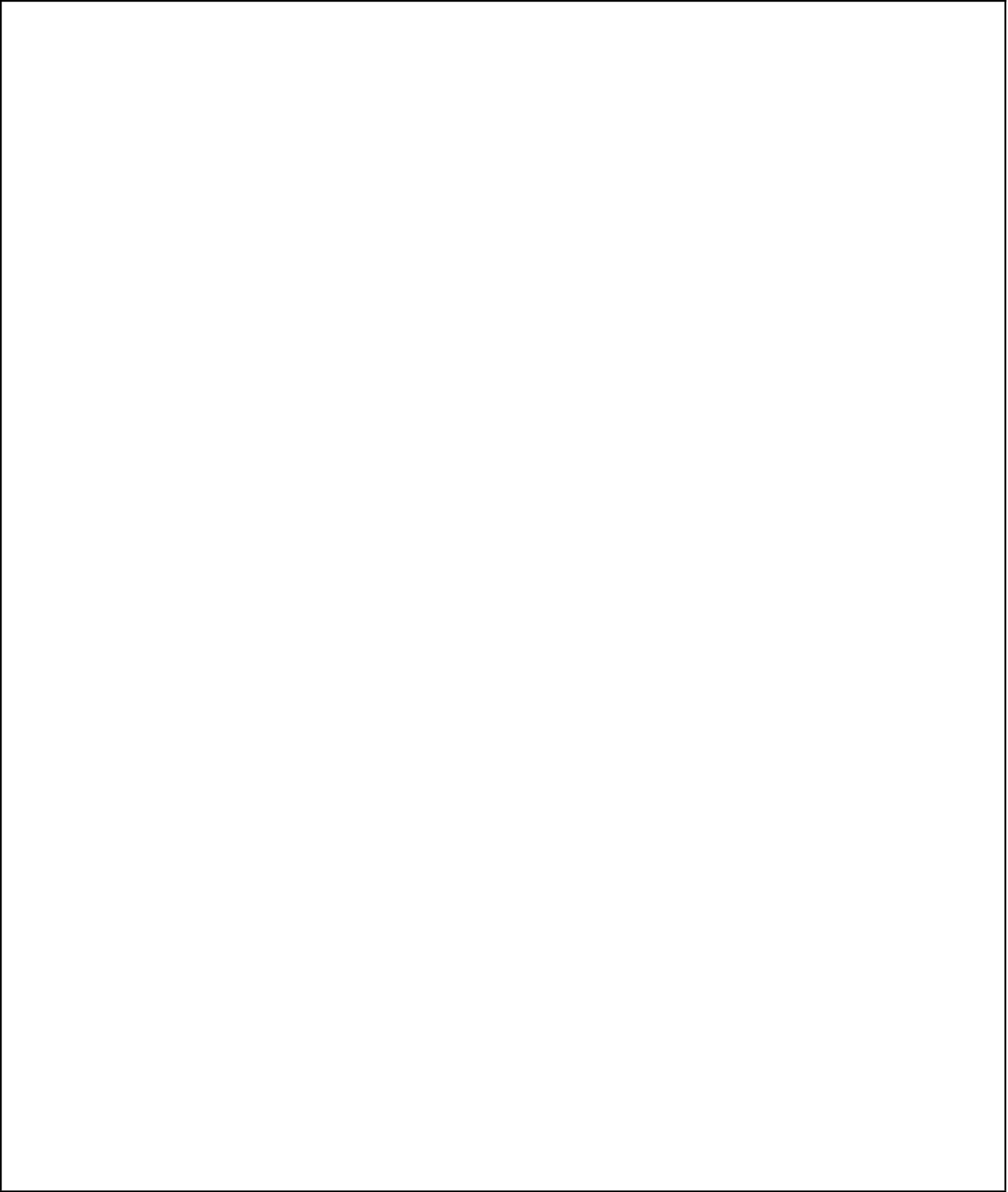
- Any factual expression in prolog is called a clause.
- There are two types of factual expressions: facts and rules
- There are three categories of statements in prolog:

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 after a query a symbol '?' statements such as:

?-parent(X,sam) Xann

?-grandfather(X,Y) X=jo, 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.

Output :

 SWI-Prolog (AMD64, Mu
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).`

`?- consult('nishanth.pl').`
true.

`?- likes(ram,What).`
`What = mango.`

`?- likes(Who,cindy).`
false.

`?- likes(Who,candy).`
`Who = bill.`

`?- red(What).`
`What = rose.`

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

`?- likes(Who,What).`
`Who = ram,`
`What = mango ;`
`Who = bill,`
`What = candy.`

DATE:	Write simple fact for the statements using prolog
EXP NO: 2	

Aim:

To write a program for simple fact statements using prolog.

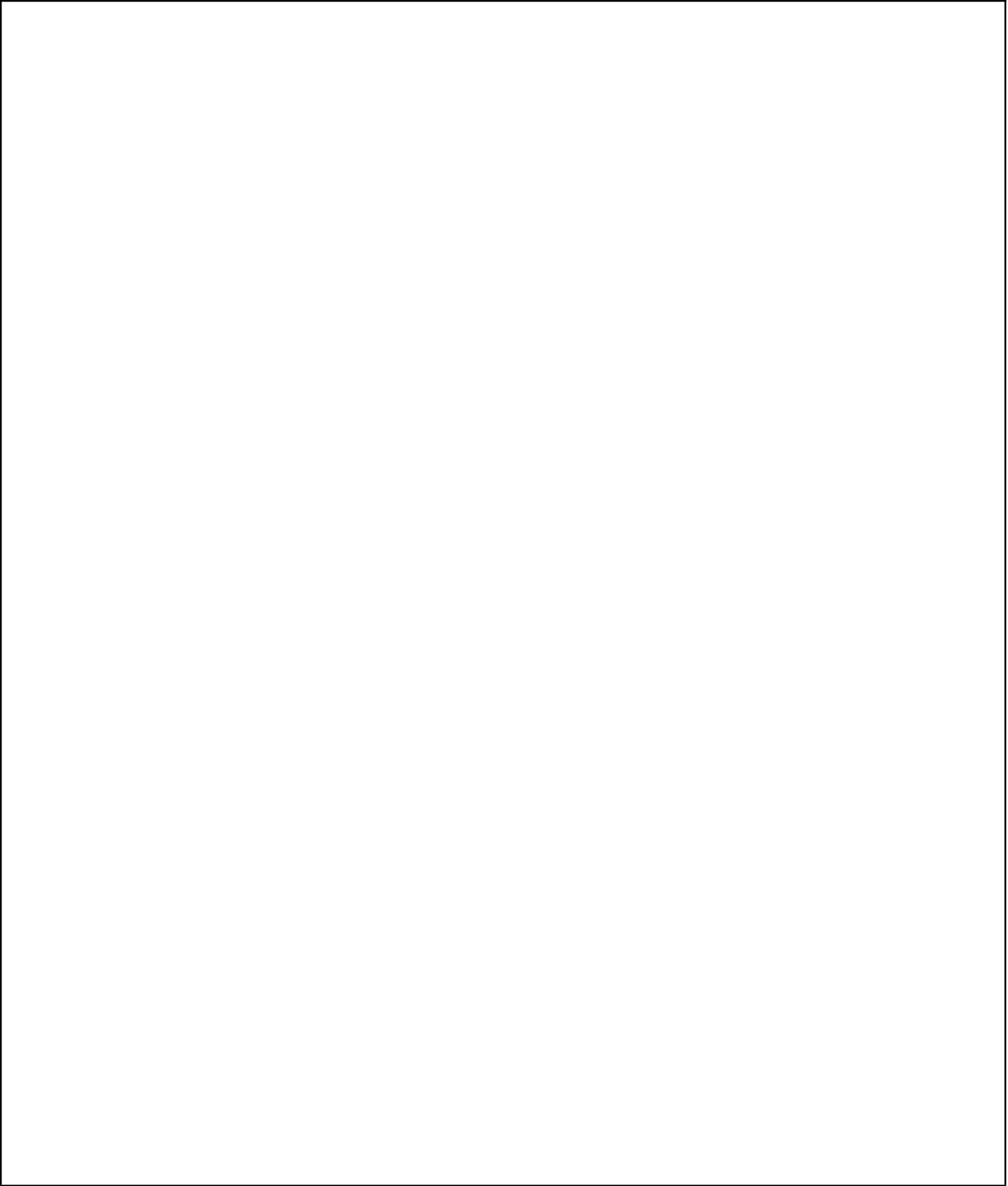
- a. Ram likes mango.
- b. Seema is a girl.
- c. Bill likes Cindy.
- d. Rose is red.
- e. John owns gold.

Program:

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

Result:

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



R

DATE:

EXP NO: 3

Write predicates one convert's centigrade temperature to Fahrenheit, other checks if a the 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:

Step 1: Start the program

Step 2: Read the input of temperature in Celsius

Step 3: $F = (9 \times C) / 5 + 32$

Step 4: Print temperature in Fahrenheit is F

Step 5: Stop the program

Rule:


Production rules:

Arithmetic:

c_to_f f is $c * 9 / 5 + 32$

freezing $f < = 32$

Output:



```
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).

?- consult('nishanth1.pl').
true.

?- c_to_f(100,X).
X = 212.

?- freezing(15).
true.

?- freezing(50).
false.

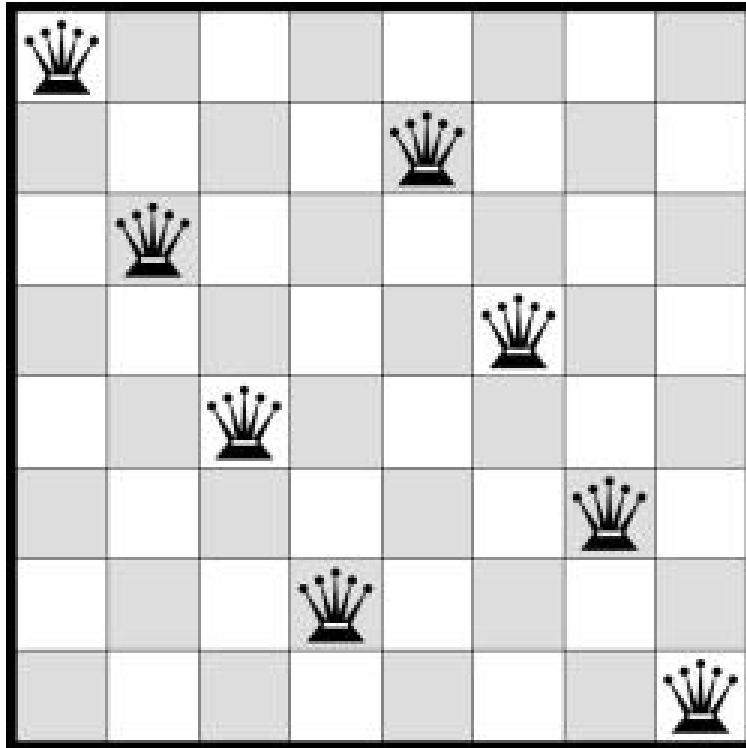
?-
```

Program:

```
c_to_f(C,F) :-  
    F is C * 9 / 5 + 32.  
freezing(F) :-  
    F =< 32.
```

Result:

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



DATE:	Write a program to solve 8-Queen problem
EXP NO: 4	

Aim:

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

Algorithm:

Step 1: Start the program from leftmost column

Step 2: If all queens are placed return true

Step 3: 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.

Step 4: If all rows have been tried and nothing worked, return false.

Step 5: Stop the program.

The 8-Queens Problem consists in placing four queens on a 8 x 8 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 8

Queens Problem: none of the 8 queens can capture each other.

Output:



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).`

`?- consult('queen.pl').`

true.

`?- solve(P).`

`P = [5, 2, 6, 1, 7, 4, 8, 3] .`

`?- setof(P,solve(P),Set),length(Set,L).`

`Set = [[1, 5, 8, 6, 3, 7, 2, 4], [1, 6, 8, 3, 7, 4, 2|...], [1, 7, 4, 6, 8, 2|...], [1, 7, 5, 8, 2|...],`
`L = 92.`

`?- █`

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([1,2,3,4,5,6,7,8],P), combine([1,2,3,4,5,6,7,8],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]).

```

Result:

Thus the program for the 8 queen problem is executed and the output is obtained successfully.

A	B	C
D	E	F
G	H	O

DATE:

Write a program to solve 8-puzzle problem

EXP NO: 5

Aim:

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

Algorithm:

Step 1: Start with the initial state and an empty move list.

Step 2: Verify if the current state is the goal state. If yes, reverse the states list to get the path and stop.

Step 3: For each possible move from the current state, generate the next state.

Step 4: Ensure the next state is not already in the current path to avoid cycles.

Step 5: Recursively apply the algorithm to the new state and update the move list.

Step 6: Once the goal is reached, display the sequence of moves and states.

Program:

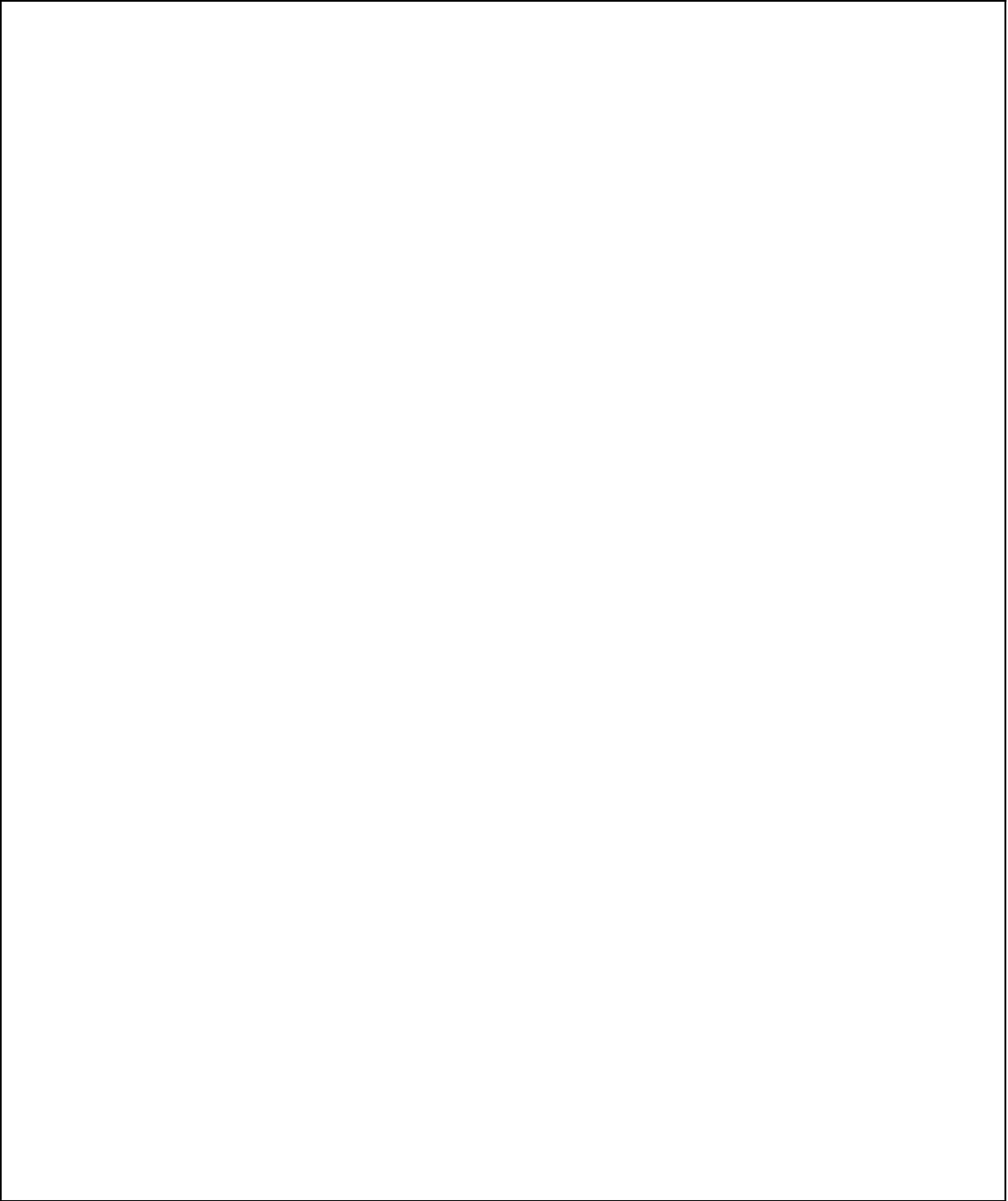
ids :-

```
start(State),  
length(Moves, N),  
dfs([State], Moves, Path), !,  
show([start|Moves], Path),  
format('~nmoves = ~w~n', [N]).
```

dfs([State|States], [], Path) :-

```
goal(State), !,  
reverse([State|States], Path).
```

dfs([State|States], [Move|Moves], Path) :-



```

move(State, Next, Move),
not(memberchk(Next, [State|States])),
dfs([Next,State|States], Moves, Path).

show([], _).
show([Move|Moves], [State|States]) :-
    State = state(A,B,C,D,E,F,G,H,I),
    format('~n~w~n~n', [Move]),
    format('~w ~w ~w~n', [A,B,C]),
    format('~w ~w ~w~n', [D,E,F]),
    format('~w ~w ~w~n', [G,H,I]),
    show(Moves, States).

% Empty position is marked with '*'

start( state(6,1,3,4,*,5,7,2,0) ).

goal( state(*,0,1,2,3,4,5,6,7) ).

move( state(*,B,C,D,E,F,G,H,J), state(B,*,C,D,E,F,G,H,J), right).
move( state(*,B,C,D,E,F,G,H,J), state(D,B,C,*,E,F,G,H,J), down ).
move( state(A,*,C,D,E,F,G,H,J), state(*,A,C,D,E,F,G,H,J), left ).
move( state(A,*,C,D,E,F,G,H,J), state(A,C,*,D,E,F,G,H,J), right).
move( state(A,*,C,D,E,F,G,H,J), state(A,E,C,D,*,F,G,H,J), down ).
move( state(A,B,*,D,E,F,G,H,J), state(A,*,B,D,E,F,G,H,J), left ).
move( state(A,B,*,D,E,F,G,H,J), state(A,B,F,D,E,*,G,H,J), down ).
move( state(A,B,C,*,E,F,G,H,J), state(*,B,C,A,E,F,G,H,J), up ).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,E,*,F,G,H,J), right).
move( state(A,B,C,*,E,F,G,H,J), state(A,B,C,G,E,F,*,H,J), down ).
move( state(A,B,C,D,*,F,G,H,J), state(A,*,C,D,B,F,G,H,J), up ).
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,F,*,G,H,J), right).

```

```
move( state(A,B,C,D,*,F,G,H,J), state(A,B,C,D,H,F,G,*,J), down );
```

Output:

```
down
```

```
left
```

```
* 0 1  
2 3 4  
5 6 7
```

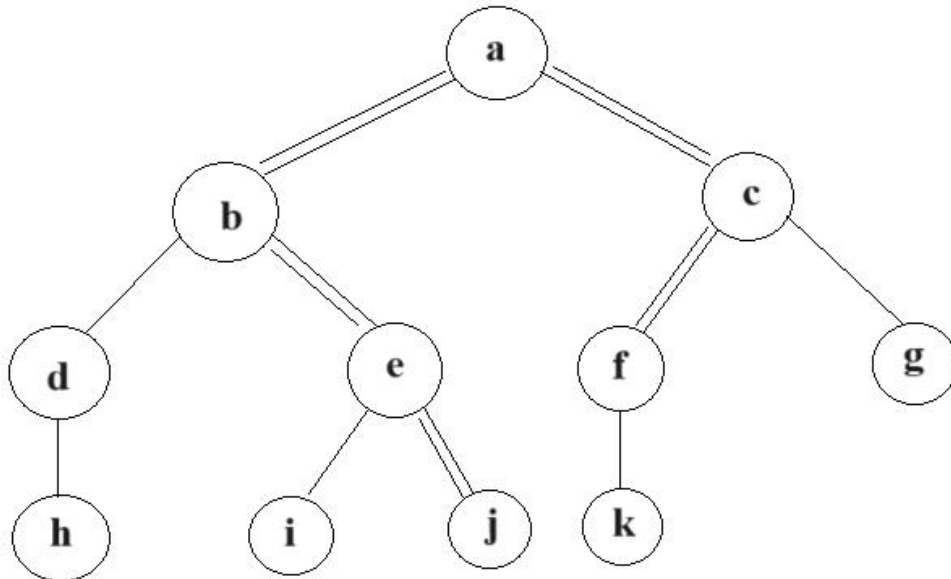
```
moves = 26
```

```
% 122,778,939 inferences, 32.641 CPU in 32.864 seconds (99% CPU, 3761538 Lips)  
true.
```

move(state(A,B,C,D,*,F,G,H,J), state(A,B,C,*,D,F,G,H,J), left).
move(state(A,B,C,D,E,*,G,H,J), state(A,B,*,D,E,C,G,H,J), up).
move(state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,*,E,G,H,J), left).
move(state(A,B,C,D,E,*,G,H,J), state(A,B,C,D,E,J,G,H,*), down).
move(state(A,B,C,D,E,F,*,H,J), state(A,B,C,D,E,F,H,*,J), right).
move(state(A,B,C,D,E,F,*,H,J), state(A,B,C,*,E,F,D,H,J), up).
move(state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,*,G,J), left).
move(state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,*,F,G,E,J), up).
move(state(A,B,C,D,E,F,G,*,J), state(A,B,C,D,E,F,G,J,*), right).
move(state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,*,G,H,F), up).
move(state(A,B,C,D,E,F,G,H,*), state(A,B,C,D,E,F,G,*,H), left).

Result:

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



DATE:

EXP NO: 6

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).

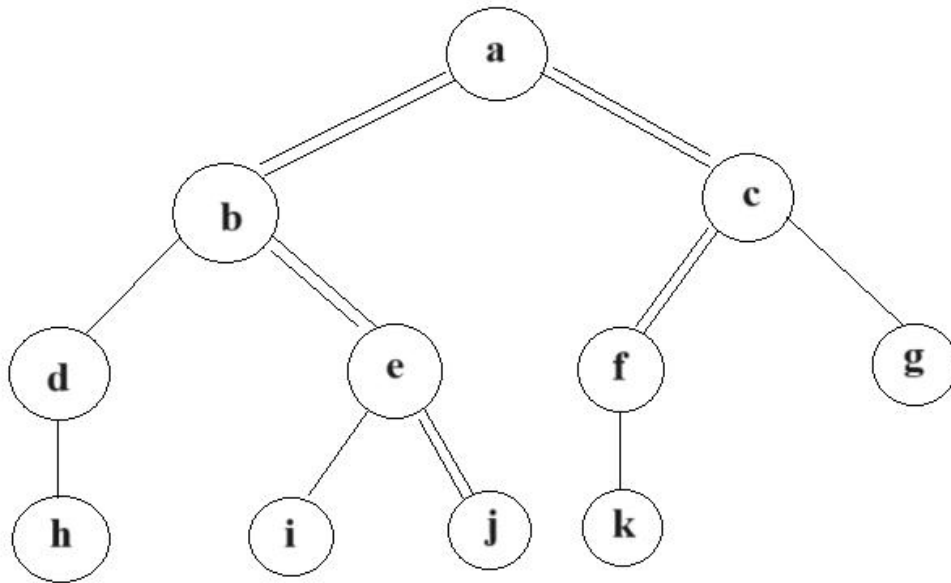
goal(j).
solve(Start,Solution):-
 bfs([[Start]],Solution).

Output:

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

Result:

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



DATE:

EXP NO: 7

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 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).
```

Output:



```
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).  
  
?- consult('dfs.pl').  
true.  
  
?- solve(a,S).  
S = [j, e, b, a] ;  
S = [f, c, a]
```

```
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)).
```

Result:

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

	Q1		
			Q2
Q3			
		Q4	

DATE:	Write a program to solve 4-Queen problem
EXP NO: 8	

Aim:

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

Algorithm:

Step 1: Start the program from leftmost column

Step 2: If all queens are placed return true

Step 3: 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.

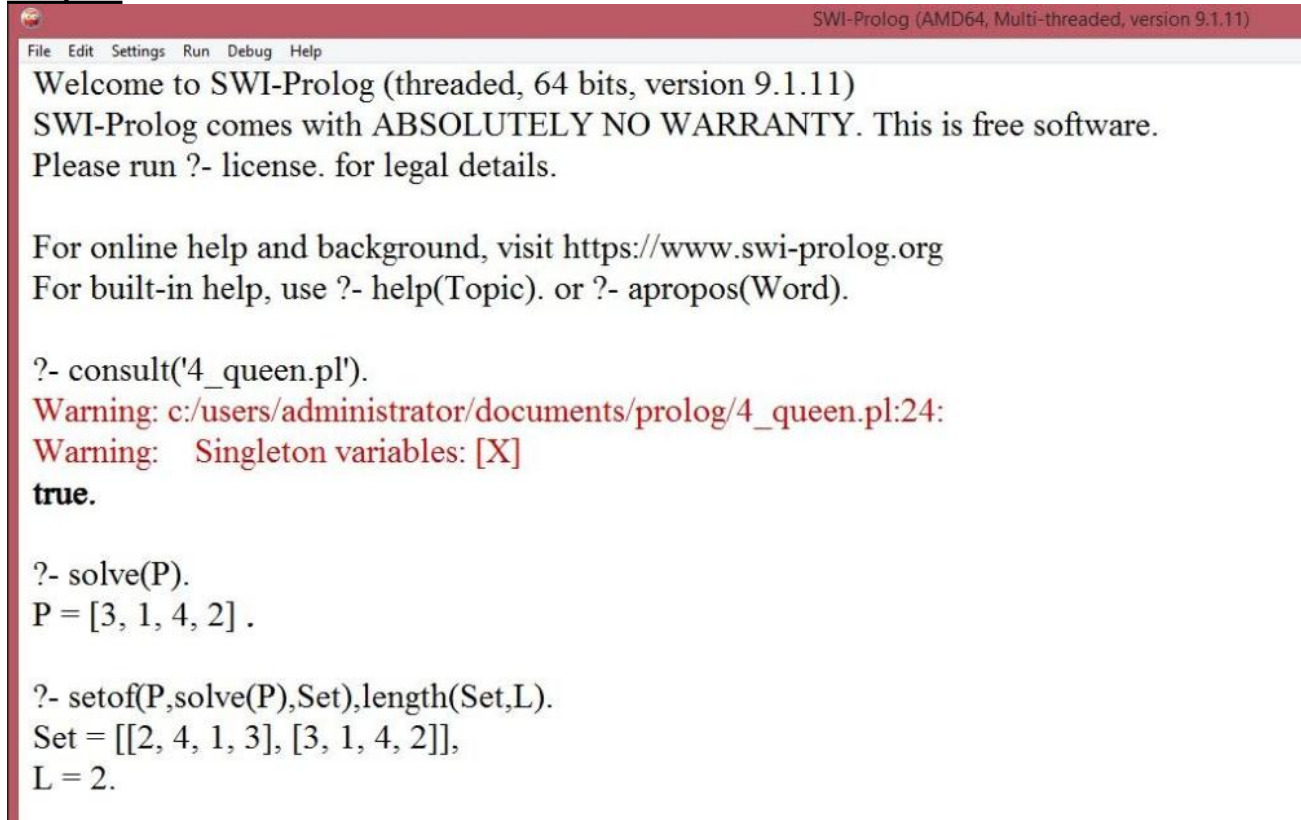
Step 4: If all rows have been tried and nothing worked, return false.

Step 5: 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.

Output:



The screenshot shows the SWI-Prolog IDE interface. The title bar reads "SWI-Prolog (AMD64, Multi-threaded, version 9.1.11)". The menu bar includes "File", "Edit", "Settings", "Run", "Debug", and "Help". The main text area displays the following output:

```
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).

?- consult('4_queen.pl').
Warning: c:/users/administrator/documents/prolog/4_queen.pl:24:
Warning: Singleton variables: [X]
true.

?- solve(P).
P = [3, 1, 4, 2] .

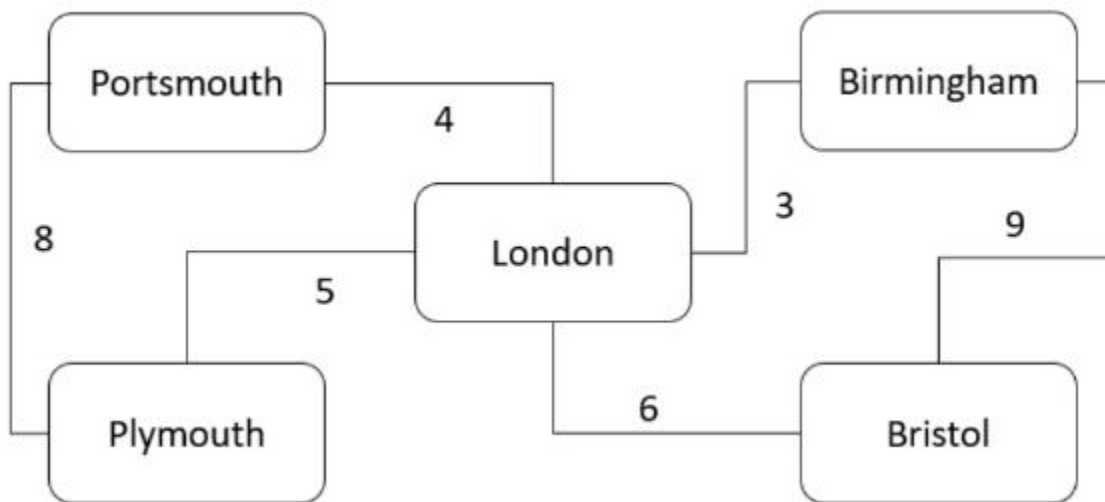
?- setof(P,solve(P),Set),length(Set,L).
Set = [[2, 4, 1, 3], [3, 1, 4, 2]],
L = 2.
```

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

Result:

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



DATE:

Write a program to solve Travelling Salesman Problem

EXP NO: 9

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).
```

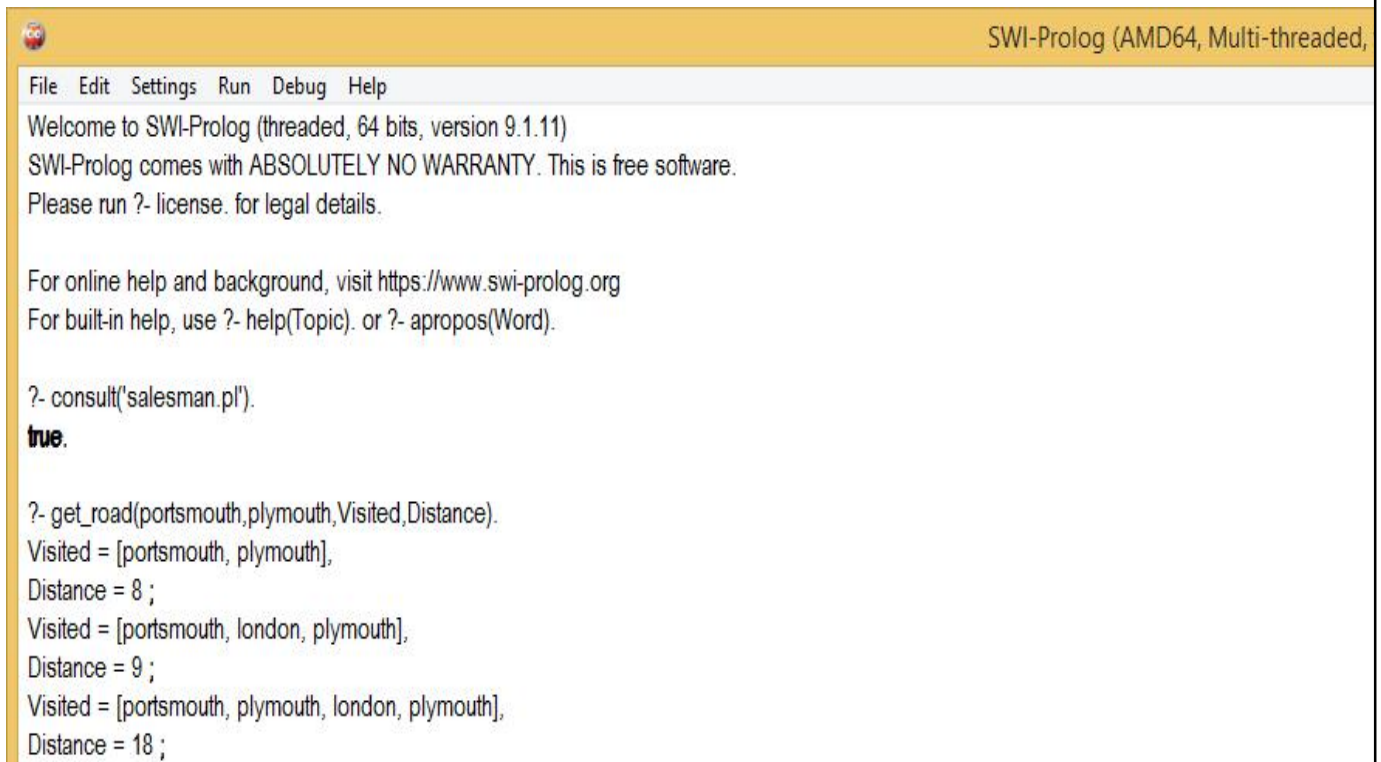
```
get_road(Start,End,Waypoints,DistanceAcc,Visited,TotalDistance):-
```

```
    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:



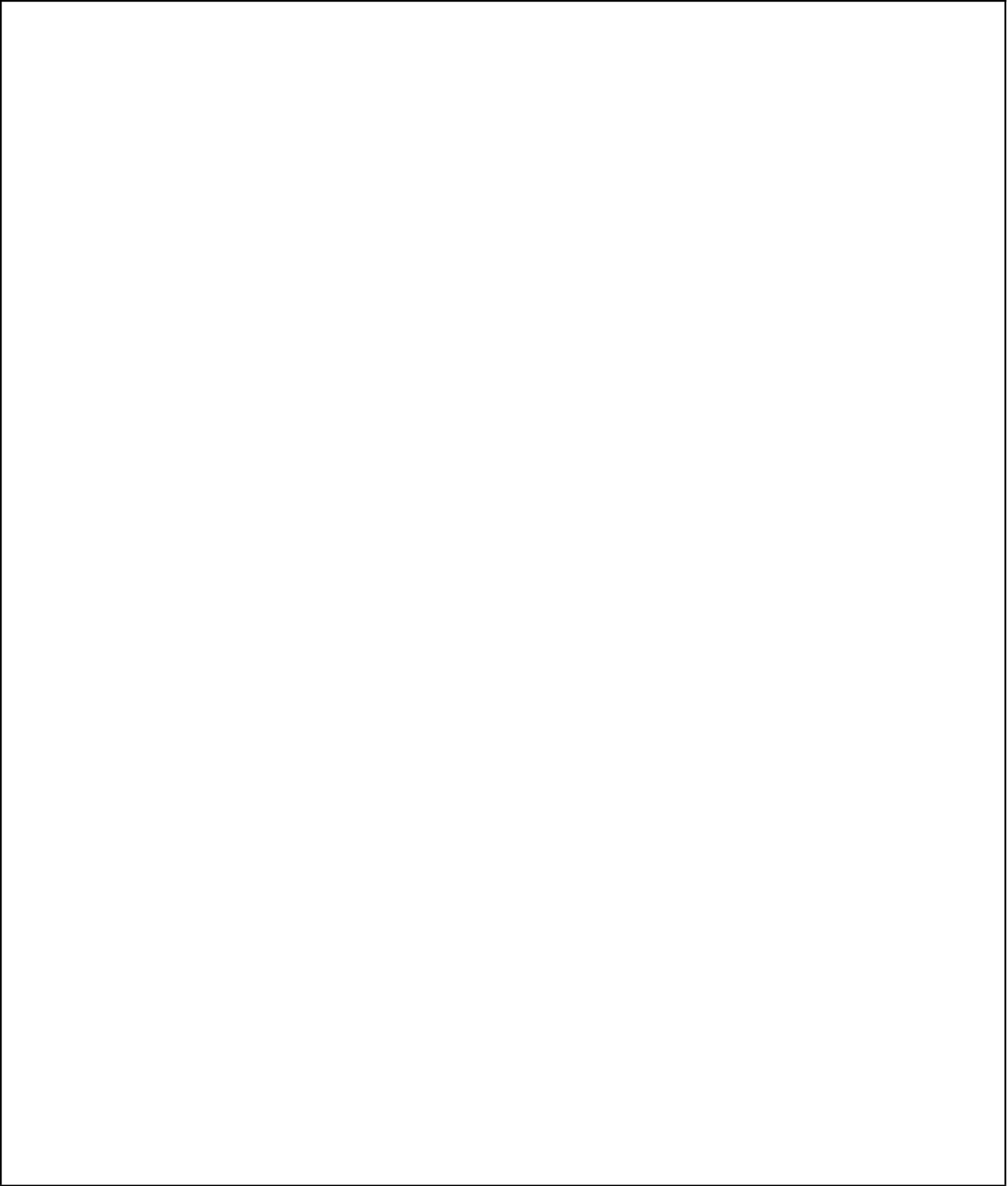
The screenshot shows the SWI-Prolog IDE window. The title bar reads "SWI-Prolog (AMD64, Multi-threaded;". The menu bar includes "File", "Edit", "Settings", "Run", "Debug", and "Help". The main text area displays the following output:

```
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).

?- consult('salesman.pl').
true.

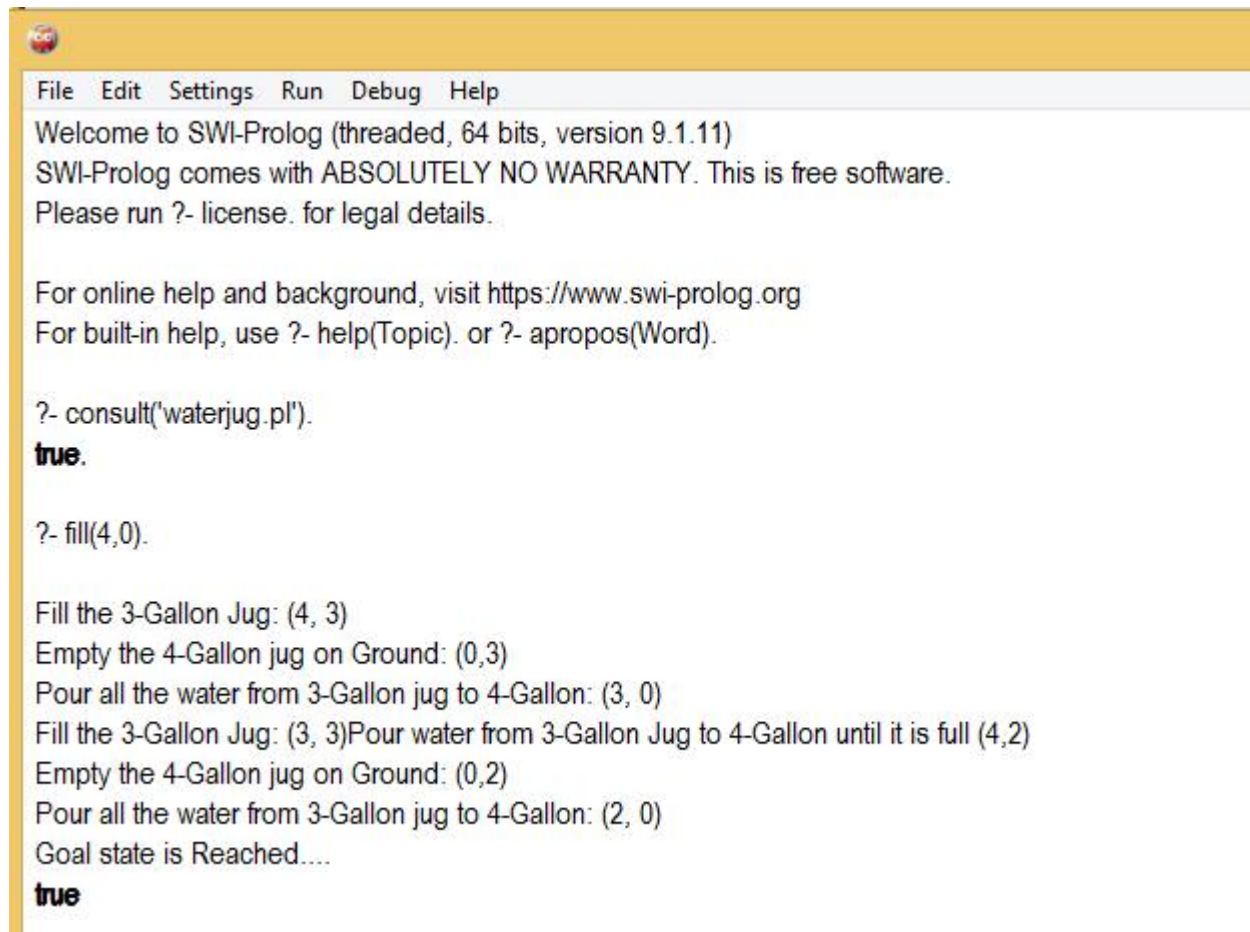
?- get_road(portsmouth,plymouth,Visited,Distance).
Visited = [portsmouth, plymouth],
Distance = 8 ;
Visited = [portsmouth, london, plymouth],
Distance = 9 ;
Visited = [portsmouth, plymouth, london, plymouth],
Distance = 18 ;
```



Result:

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

Output:



```
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).

?- consult('waterjug.pl').
true.

?- fill(4,0).

Fill the 3-Gallon Jug: (4, 3)
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 it is full (4,2)
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
```

DATE:

Write a prolog program to solve water jug problem

EXP NO: 10

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 5 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),  
write('Pour water from 3-Gallon Jug to 4-Gallon until it 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 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 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 to 3-Gallon: (0, ', write(Y1), write(')'), fill(0, Y1).
```

```
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.
```

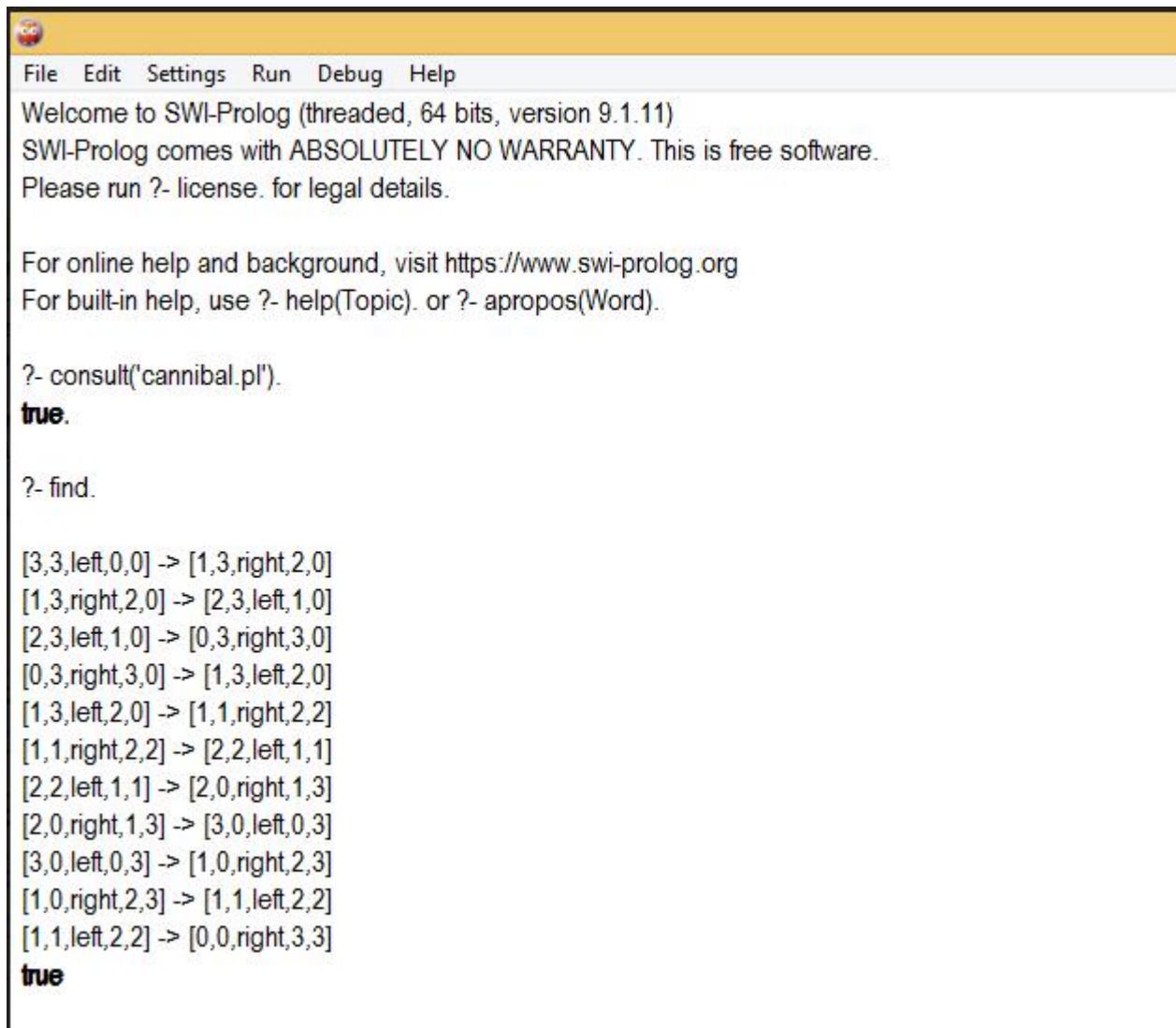
```
fill(X, Y) :-  
    X < 4, Y > 3,  
    write('3L Jug Overflowed'), nl.
```

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

Result:

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

Output:



```
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).

?- consult('cannibal.pl').
true.

?- 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
```

DATE:

Write a program to solve missionaries and cannibal problem

EXP NO: 11

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]],_).
```

Result:

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

Output:



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).`

`?- consult('library.pl').`

true.

`?- taken('1984', Student, TakenDate, ReturnDate).`

Student = 'Bob',

TakenDate = 2023-10-20,

ReturnDate = 2023-11-5.

`?- author(Author, 'The Hobbit').`

Author = 'J.R.R. Tolkien'.

`?- available('The Hobbit', Available).`

Available = 1.

`?- in_stock('To kill a Mockingbird', InStock).`

InStock = 5.

`?-`

DATE:

Library Management System

EXP NO: 12

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, student's 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:

```
book('The Hobbit','J.R.R Tolkien',1937,2).  
book('1984','George Orwell',1949,3).  
book('To kill a Mockingbird','Harper Lee',1960,5).  
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','Home',-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('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).

```

```

author(Author,Title):-
    book(Title,Author,_,_).
in_stock(Title,InStock):-
    book(Title,_,_,InStock).

```

```

available(Title,Available):-
    book(Title,_,_,Total),
    findall(Student,taken(Title,Student,_,_),Taken),
    length(Taken,TakenCount),
    Available is Total - TakenCount.

```

Result:

Thus, the Library Management system is simulated using Prolog.

Output:

```
Python 3.7.17 Shell

File Edit Shell Debug Options Window Help

Python 3.7.17 (default, Apr 27 2024, 21:22:13)
[GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /home/nishanth-s-pradeep/Desktop/myenv/ailab.py =====
List Trainer: [#          ] 7%List Trainer: [###          ] 14%List Trainer: [###
9%List Trainer: [#####          ] 36%List Trainer: [#####          ] 43%List Trainer: [#####
] 57%List Trainer: [#####          ] 64%List Trainer: [#####          ] 71%List Trainer: [#####
] 86%List Trainer: [#####          ] 93%List Trainer: [#####          ] 100%
User: hi
Bot: Hi there
User: what's your name
Bot: I'm a chatbot
User: what is your age
Bot: I'm ageless!
User: what is your favourite food
Bot: I don't eat
User: you have iphone
Bot: I've everything!
User: what is your job
```

DATE:

Simple ChatBot using Python

EXP NO: 13

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: ")`.

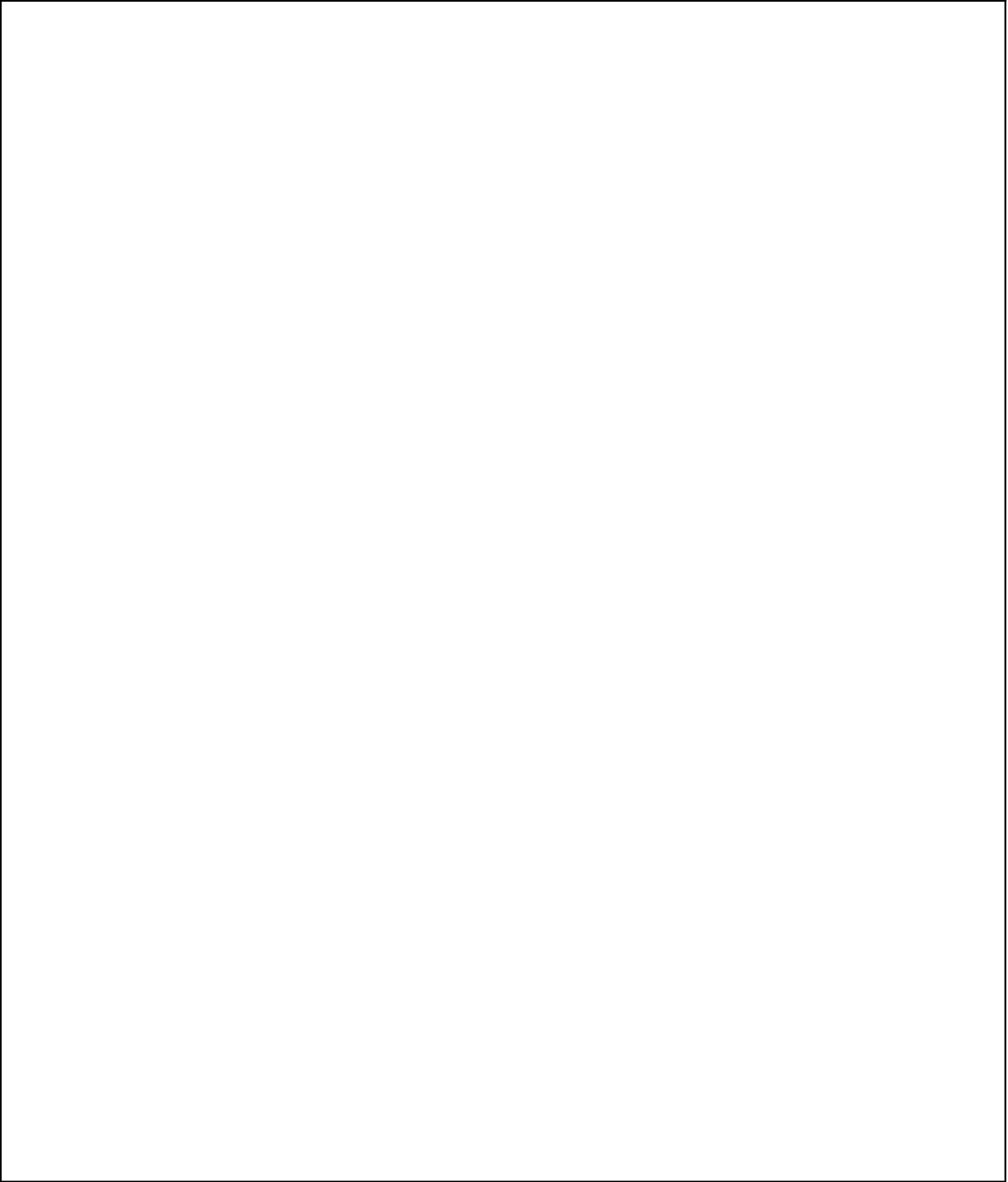
Step 8: Check for the "stop" condition; if input is "stop", break the loop.

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)))
```

Result:

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