INDEX

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sr No. | Date | Title | Page No. | Grade | Sign |
| 1 |  | Study of PROLOG environment with simple programs. | 1 |  |  |
| 2 |  | Learn Backtracking and Unification. Implement Medical diagnosis system with PROLOG. | 3 |  |  |
| 3 |  | Learn Different Predicates. And implement revised medical diagnosis system. | 6 |  |  |
| 4 |  | Learn Recursion and Implement it with examples. | 10 |  |  |
| 5 |  | Learn CUT predicate, Arithmetic predicates and implement with examples. | 11 |  |  |
| 6 |  | Study and implementation of compound Objects and dynamic database. | 13 |  |  |
| 7 |  | Study and implementation of Lists and strings. | 15 |  |  |
| 8 |  | Write a program to implement Tic-tac-toe game problem. | 18 |  |  |
| 9 |  | Write a program to implement BFS and DFS. | 22 |  |  |
| 10 |  | Write a program to implement Single Player Game (Using Heuristic Function). | 25 |  |  |
| 11 |  | Write a program to implement A\* algorithm. | 28 |  |  |
| 12 |  | N-Queens problem. | 33 |  |  |
| 13 |  | 8-puzzle problem. | 35 |  |  |
| 14 |  | Travelling salesman problem. | 38 |  |  |
| 15 |  | Convert following Prolog predicates into Semantic Net. | 40 |  |  |
| 16 |  | Write the Conceptual Dependency for following statements. (a) John gives Mary a book. (b) John gave Mary the book yesterday. | 41 |  |  |
|  |  | Beyond Syllabus Practical | 42 |  |  |

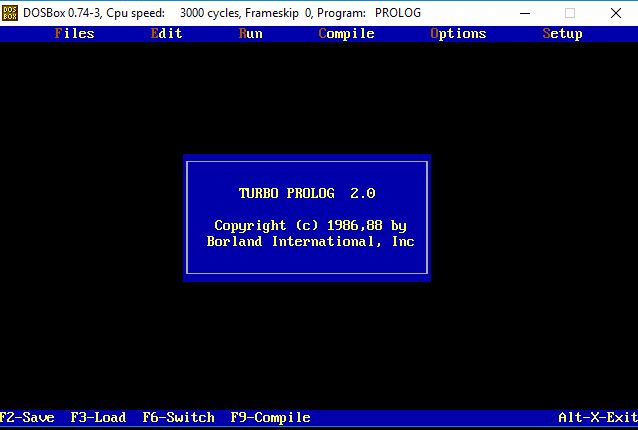
# Practical – 1

# Aim: Study of PROLOG environment with simple programs.

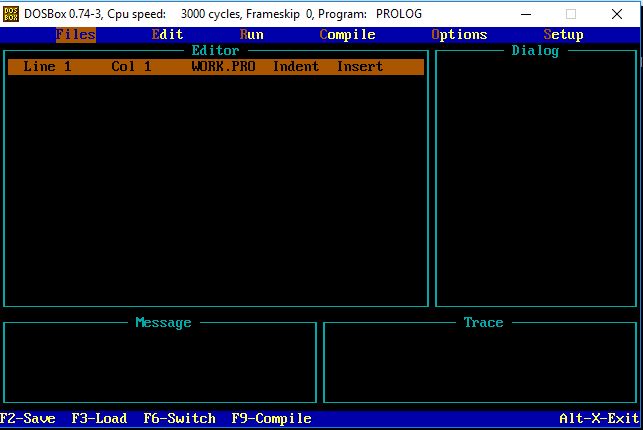
## Prolog Environment: -

Prolog is a very important tool in artificial intelligence applications programming and in the development of expert systems. Several well-known expert system shells are written in Prolog, including APES, ESP/Advisor and Xi.

Once you have a copy of the system on your working disk and you are in the appropriate directory, type PROLOG. You should see the logon message shown in Figure.



Now press the space bar and the Turbo Prolog main menu and four system windows will appear as shown in Figure.



## Sample Programs: -

Prac\_1\_1.pro

predicates

like(symbol,symbol,symbol)

clauses

like(nidhi,dove,descendents).

like(bhumi,scarjo,avengers).

Prac\_1\_2.pro

domains

brand, color = symbol

age = integer

price, mileage = real

predicates

car(brand,mileage,age,color,price)

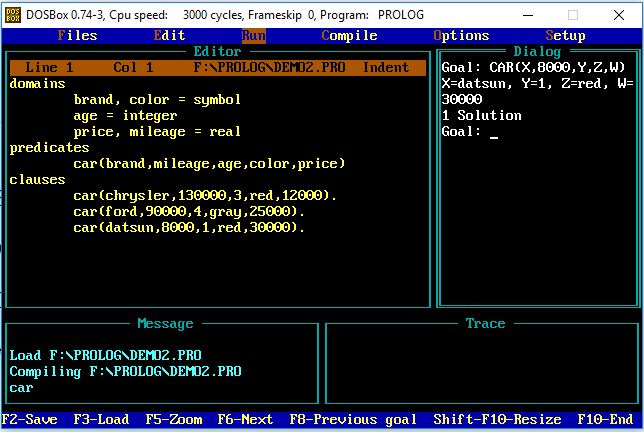
clauses

car(chrysler,130000,3,red,12000).

car(ford,90000,4,gray,25000).

car(datsun,8000,1,red,30000).

## Outputs:



# Practical – 2

# Aim: Learn Backtracking and Unification. Implement Medical diagnosis system with PROLOG.

## Program for backtracking: -

predicates

can\_buy(symbol,symbol,symbol)

in\_form(symbol,symbol)

avail(symbol)

clauses

can\_buy(x,y,z):-

in\_form(y,z),

avail(z),

can\_buy(nidhi,bluray,avengers).

can\_buy(bhumi,dvd,frozenii).

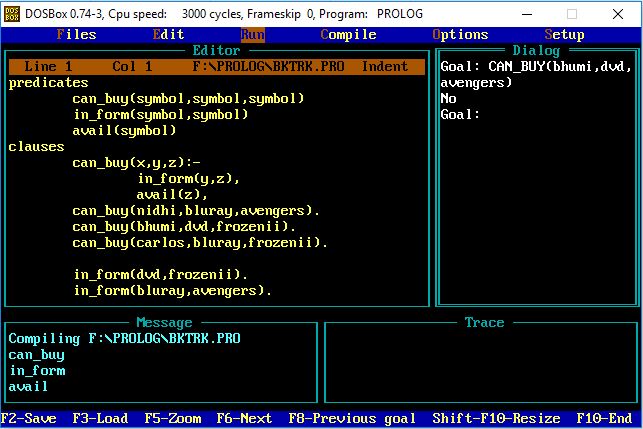
can\_buy(carlos,bluray,frozenii).

in\_form(dvd,frozenii).

in\_form(bluray,avengers).

avail(avengers).

## Output: -



## Program for medical diagnosis: -

domains

disease,indication,name = symbol

predicates

hypothesis(name,disease)

symptom(name,indication)

clauses

symptom(amit,fever).

symptom(amit,rash).

symptom(amit,headache).

symptom(amit,runn\_nose).

symptom(kaushal,chills).

symptom(kaushal,fever).

symptom(kaushal,hedache).

symptom(dipen,runny\_nose).

symptom(dipen,rash).

symptom(dipen,flu).

hypothesis(Patient,measels):-

symptom(Patient,fever),

symptom(Patient,cough),

symptom(Patient,conjunctivitis),

symptom(Patient,rash).

hypothesis(Patient,german\_measles) :-

symptom(Patient,fever),

symptom(Patient,headache),

symptom(Patient,runny\_nose),

symptom(Patient,rash).

hypothesis(Patient,flu) :-

symptom(Patient,fever),

symptom(Patient,headache),

symptom(Patient,body\_ache),

symptom(Patient,chills).

hypothesis(Patient,common\_cold) :-

symptom(Patient,headache),

symptom(Patient,sneezing),

symptom(Patient,sore\_throat),

symptom(Patient,chills),

symptom(Patient,runny\_nose).

hypothesis(Patient,mumps) :-

symptom(Patient,fever),

symptom(Patient,swollen\_glands).

hypothesis(Patient,chicken\_pox) :-

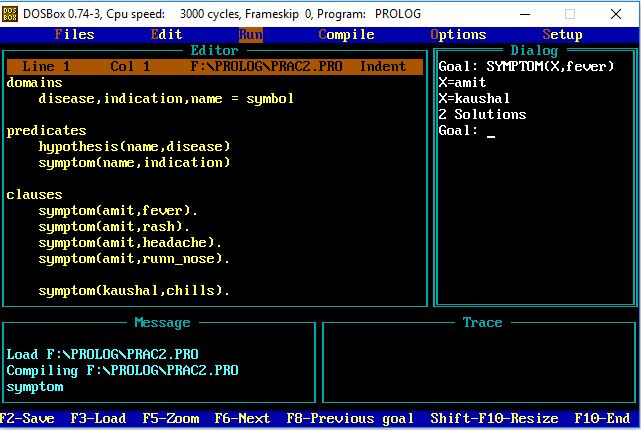
symptom(Patient,fever),

symptom(Patient,rash),

symptom(Patient,body\_ache),

symptom(Patient,chills).

## Output: -



# Practical – 3

# Aim: Learn Different Predicates. And implement revised medical diagnosis system.

## Program for medical diagnosis (revised): -

domains

disease,indication = symbol

Patient,name = string

predicates

hypothesis(string,disease)

symptom(name,indication)

response(char)

go

clauses

go :-

write("What is the patient's name?"),

readln(Patient),

hypothesis(Patient,Disease),

write(Patient,"probably has ",Disease,"."),nl.

go :-

write("Sorry, I do not seem to be able to"),nl,

write("diagnose the disease."),nl.

symptom(Patient,fever) :-

write("Does ",Patient," have a fever (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,rash) :-

write("Does ",Patient," have a rash (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,headache) :-

write("Does ",Patient," have a headache (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,runny\_nose) :-

write("Does ",Patient," have a runny\_nose (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,conjunctivitis) :-

write("Does ",Patient," have a conjunctivitis (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,cough) :-

write("Does ",Patient," have a cough (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,body\_ache) :-

write("Does ",Patient," have a body\_ache (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,chills) :-

write("Does ",Patient," have a chills (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,sore\_throat) :-

write("Does ",Patient," have a sore\_throat (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,sneezing) :-

write("Does ",Patient," have a sneezing (y/n) ?"),

response(Reply),

Reply='y'.

symptom(Patient,swollen\_glands) :-

write("Does ",Patient," have a swollen\_glands (y/n) ?"),

response(Reply),

Reply='y'.

hypothesis(Patient,measles) :-

symptom(Patient,fever),

symptom(Patient,cough),

symptom(Patient,conjunctivitis),

symptom(Patient,runny\_nose),

symptom(Patient,rash).

hypothesis(Patient,german\_measles) :-

symptom(Patient,fever),

symptom(Patient,headache),

symptom(Patient,runny\_nose),

symptom(Patient,rash).

hypothesis(Patient,flu) :-

symptom(Patient,fever),

symptom(Patient,headache),

symptom(Patient,body\_ache),

symptom(Patient,conjunctivitis),

symptom(Patient,chills),

symptom(Patient,sore\_throat),

symptom(Patient,runny\_nose),

symptom(Patient,cough).

hypothesis(Patient,common\_cold) :-

symptom(Patient,headache),

symptom(Patient,sneezing),

symptom(Patient,sore\_throat),

symptom(Patient,runny\_nose),

symptom(Patient,chills).

hypothesis(Patient,mumps) :-

symptom(Patient,fever),

symptom(Patient,swollen\_glands).

hypothesis(Patient,chicken\_pox) :-

symptom(Patient,fever),

symptom(Patient,chills),

symptom(Patient,body\_ache),

symptom(Patient,rash).

hypothesis(Patient,measles) :-

symptom(Patient,cough),

symptom(Patient,sneezing),

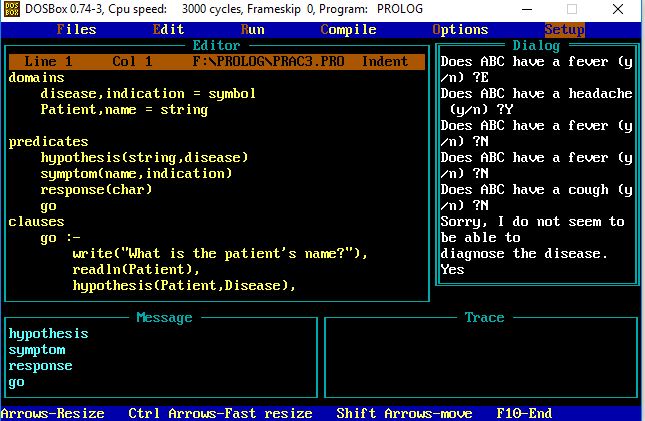
symptom(Patient,runny\_nose).

response(Reply) :-

readchar(Reply),

write(Reply),nl.

## Output: -



# Practical – 4

# Aim: Learn Recursion and Implement it with examples.

## Program for finding factorial of number using recursion: -

predicates

start

find\_factorial(real,real)

goal

clearwindow,

start.

clauses

start:-

write("Enter non negative number = "),

readreal(Num),

Result = 1.0,

find\_factorial(Num,Result).

find\_factorial(Num,Result):-

Num <> 0,

NewResult = Num \* Result,

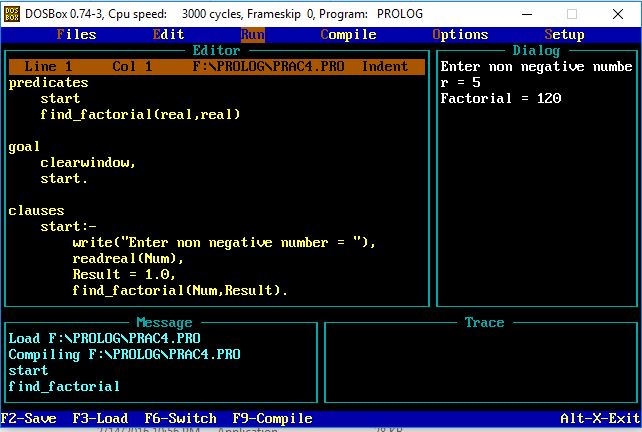
NewNum = Num - 1,

find\_factorial(NewNum,NewResult).

find\_factorial(\_,Result):-

write("Factorial = ",Result),nl.

## Output: -



# Practical – 5

# Aim: Learn CUT predicate, Arithmetic predicates and implement with examples.

## Program for CUT predicate: -

domains

name,sex,interest = symbol

interests = interest\*

predicates

findpairs

person(name,sex,interests)

member(interest,interests)

common\_interest(interests, interests, interest)

clauses

findpairs if person(Man, m, ILIST1 ) and

person( Woman, f, ILIST2 ) and

common\_interest( ILIST1, ILIST2,\_) and

write( Man, "might like",Woman) and nl and

fail.

findpairs:- write ("-----------end of the 1ist---" ).

common\_interest( IL1, IL2, X ) if

member(X, IL1 ) and member(X, IL2) and !.

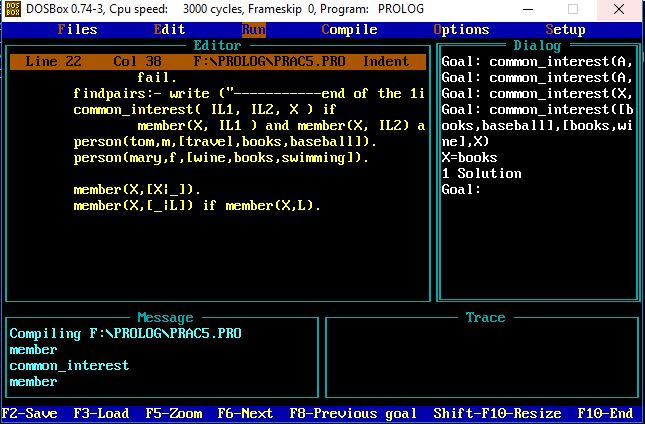
person(tom,m,[travel,books,baseball]).

person(mary,f,[wine,books,swimming]).

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

member(X,[\_|L]) if member(X,L).

## Output: -



## Program for arithmetic predicates: -

domains

d = pair(integer,integer) ; single(integer);none

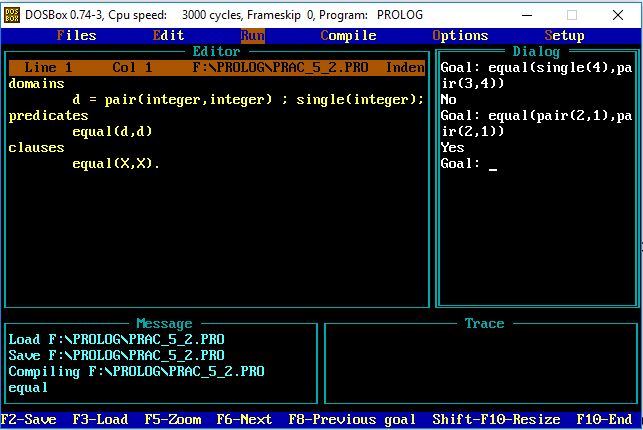
predicates

equal(d,d)

clauses

equal(X,X).

## Output: -



# Practical – 6

# Aim: Study and implementation of compound Objects and dynamic database.

## Program for compound objects: -

domains

row, column, step = integer

movement = up(step); down(step);

left(step) ; right(step)

predicates

move\_cursor(row,column,movement)

clauses

move\_cursor(R,C,up(Step)) :-

Rl= R-Step,cursor(Rl,C).

move\_cursor(R,C,down(Step)) :-

Rl= R+Step,cursor(Rl,C).

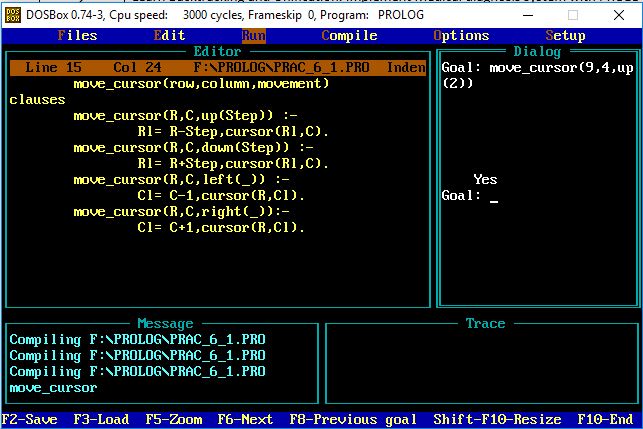
move\_cursor(R,C,left(\_)) :-

Cl= C-1,cursor(R,Cl).

move\_cursor(R,C,right(\_)):-

Cl= C+1,cursor(R,Cl).

## Output: -



## Program for dynamic database: -

domains

name,addr = string

one\_data\_record = p(name,addr)

file = file\_of\_data\_records

predicates

person(name,addr)

moredata(file)

clauses

person(Name,Addr):-

openread(file\_of\_data\_records,"dd.dat") ,

readdevice(file\_of\_data\_records),

moredata(file\_of\_data\_records),

readterl(one\_data\_record,p(Name,Addr)).

moredata(\_) .

moredata(File):- not(eof(File)),moredata(File).

Provided the file dd.dat contains facts belonging to the description domain, such as

p("Peter" ,"28th Street" )

p("Curt","Wall Street" )

Output: -

Goal: person("Peter",Address).

Address="28th Street"

1 Solution

Goal: person("Peter","Not an address").

False

Goal : ...

# Practical – 7

# Aim: Study and implementation of Lists and strings.

## Program for lists: -

List Membership

list\_member(X, [X|\_]).

list\_member(X, [\_|Tail]):-

list\_member(X,Tail).

Concatenation

concatenation([],L,L).

concatenation([X1|L1],L2,[X1|L3]):-

concatenation(L1,L2,L3).

Append

list\_member(X,[X|\_]).

list\_member(X,[\_|Tail]):-

list\_member(X,Tail).

list\_append(A,T,T):-

list\_member(A,T),!.

list\_append(A,Tail,[A|Tail]).

## Program:

list\_member(b,[a,b,c]).

list\_member([b,c],[a,[b,c]]).

list\_member(X,[a,b,c,d,e,f]).

concatenation([a,b,c],[d,e,f,g],L).

L= [a,b,c,d,e,f,g]

concatenation([a,b,c],L,[a,b,c,d,e,f,g,h]).

concatenation([],L,[a,b,c,d,e,f,g,h]).

concatenation([a,b,c],[],L).

concatenation([],[],L).

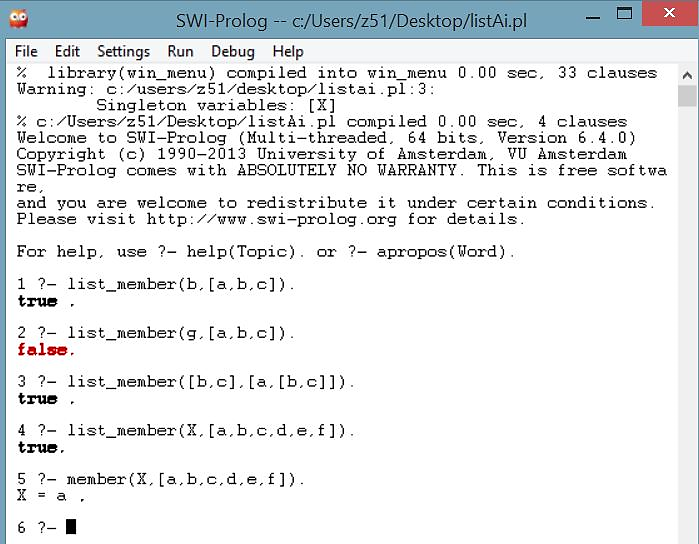
list\_append([1,2,3],[4,5,6],X).

X=[[1,2,3],4,5,6]

list\_append(3,[4,5,6],X).

X=[3,4,5,6]

## Output: -



## Program for strings: -

domains

charlist=char\*

predicates

string\_chlist(string,charlist)

clauses

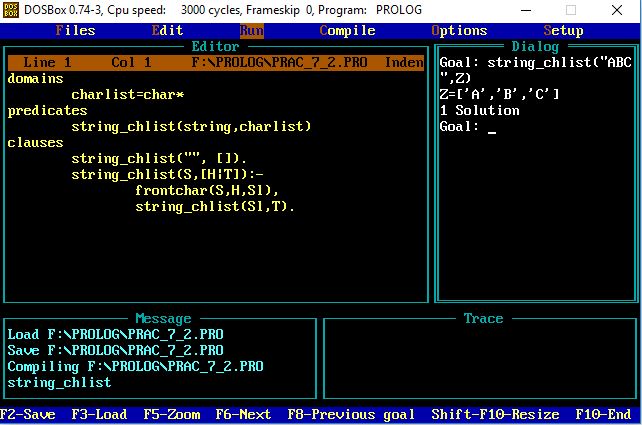
string\_chlist("", []).

string\_chlist(S,[H|T]):-

frontchar(S,H,Sl),

string\_chlist(Sl,T).

## Output: -



# Practical – 8

# Aim: Write a program to implement Tic-tac-toe game problem.

## Program: -

% A Tic-Tac-Toe program in Prolog. S. Tanimoto, May 11, 2003.

% To play a game with the computer, type

% playo.

% To watch the computer play a game with itself, type

% selfgame.

% Predicates that define the winning conditions:

win(Board, Player) :- rowwin(Board, Player).

win(Board, Player) :- colwin(Board, Player).

win(Board, Player) :- diagwin(Board, Player).

rowwin(Board, Player) :- Board = [Player,Player,Player,\_,\_,\_,\_,\_,\_].

rowwin(Board, Player) :- Board = [\_,\_,\_,Player,Player,Player,\_,\_,\_].

rowwin(Board, Player) :- Board = [\_,\_,\_,\_,\_,\_,Player,Player,Player].

colwin(Board, Player) :- Board = [Player,\_,\_,Player,\_,\_,Player,\_,\_].

colwin(Board, Player) :- Board = [\_,Player,\_,\_,Player,\_,\_,Player,\_].

colwin(Board, Player) :- Board = [\_,\_,Player,\_,\_,Player,\_,\_,Player].

diagwin(Board, Player) :- Board = [Player,\_,\_,\_,Player,\_,\_,\_,Player].

diagwin(Board, Player) :- Board = [\_,\_,Player,\_,Player,\_,Player,\_,\_].

% Helping predicate for alternating play in a "self" game:

other(x,o).

other(o,x).

game(Board, Player) :- win(Board, Player), !, write([player, Player, wins]).

game(Board, Player) :-

other(Player,Otherplayer),

move(Board,Player,Newboard),

!,

display(Newboard),

game(Newboard,Otherplayer).

move([b,B,C,D,E,F,G,H,I], Player, [Player,B,C,D,E,F,G,H,I]).

move([A,b,C,D,E,F,G,H,I], Player, [A,Player,C,D,E,F,G,H,I]).

move([A,B,b,D,E,F,G,H,I], Player, [A,B,Player,D,E,F,G,H,I]).

move([A,B,C,b,E,F,G,H,I], Player, [A,B,C,Player,E,F,G,H,I]).

move([A,B,C,D,b,F,G,H,I], Player, [A,B,C,D,Player,F,G,H,I]).

move([A,B,C,D,E,b,G,H,I], Player, [A,B,C,D,E,Player,G,H,I]).

move([A,B,C,D,E,F,b,H,I], Player, [A,B,C,D,E,F,Player,H,I]).

move([A,B,C,D,E,F,G,b,I], Player, [A,B,C,D,E,F,G,Player,I]).

move([A,B,C,D,E,F,G,H,b], Player, [A,B,C,D,E,F,G,H,Player]).

display([A,B,C,D,E,F,G,H,I]) :- write([A,B,C]),nl,write([D,E,F]),nl,

write([G,H,I]),nl,nl.

selfgame :- game([b,b,b,b,b,b,b,b,b],x).

% Predicates to support playing a game with the user:

x\_can\_win\_in\_one(Board) :- move(Board, x, Newboard), win(Newboard, x).

% The predicate orespond generates the computer's (playing o) reponse

% from the current Board.

orespond(Board,Newboard) :-

move(Board, o, Newboard),

win(Newboard, o),

!.

orespond(Board,Newboard) :-

move(Board, o, Newboard),

not(x\_can\_win\_in\_one(Newboard)).

orespond(Board,Newboard) :-

move(Board, o, Newboard).

orespond(Board,Newboard) :-

not(member(b,Board)),

!,

write('Cats game!'), nl,

Newboard = Board.

% The following translates from an integer description

% of x's move to a board transformation.

xmove([b,B,C,D,E,F,G,H,I], 1, [x,B,C,D,E,F,G,H,I]).

xmove([A,b,C,D,E,F,G,H,I], 2, [A,x,C,D,E,F,G,H,I]).

xmove([A,B,b,D,E,F,G,H,I], 3, [A,B,x,D,E,F,G,H,I]).

xmove([A,B,C,b,E,F,G,H,I], 4, [A,B,C,x,E,F,G,H,I]).

xmove([A,B,C,D,b,F,G,H,I], 5, [A,B,C,D,x,F,G,H,I]).

xmove([A,B,C,D,E,b,G,H,I], 6, [A,B,C,D,E,x,G,H,I]).

xmove([A,B,C,D,E,F,b,H,I], 7, [A,B,C,D,E,F,x,H,I]).

xmove([A,B,C,D,E,F,G,b,I], 8, [A,B,C,D,E,F,G,x,I]).

xmove([A,B,C,D,E,F,G,H,b], 9, [A,B,C,D,E,F,G,H,x]).

xmove(Board, N, Board) :- write('Illegal move.'), nl.

% The 0-place predicate playo starts a game with the user.

playo :- explain, playfrom([b,b,b,b,b,b,b,b,b]).

explain :-

write('You play X by entering integer positions followed by a period.'),

nl,

display([1,2,3,4,5,6,7,8,9]).

playfrom(Board) :- win(Board, x), write('You win!').

playfrom(Board) :- win(Board, o), write('I win!').

playfrom(Board) :- read(N),

xmove(Board, N, Newboard),

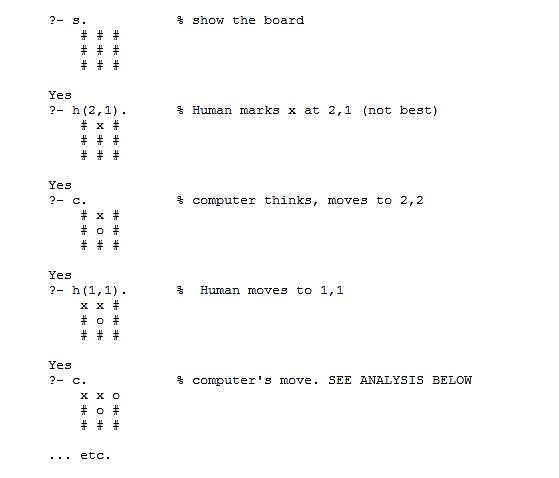
display(Newboard),

orespond(Newboard, Newnewboard),

display(Newnewboard),

playfrom(Newnewboard).

## Output: -



# Practical – 9

# Aim: Write a program to implement BFS and DFS.

## Prolog program to solve the water-jug puzzle using BFS: -

database

visited\_state(integer,integer)

predicates

state(integer,integer)

clauses

state(2,0).

state(X,Y):- X < 4,

not(visited\_state(4,Y)),

assert(visited\_state(X,Y)),

write("Fill the 4-Gallon Jug: (",X,",",Y,") --> (", 4,",",Y,")\n"),

state(4,Y).

state(X,Y):- Y < 3,

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Fill the 3-Gallon Jug: (", X,",",Y,") --> (", X,",",3,")\n"),

state(X,3).

state(X,Y):- X > 0,

not(visited\_state(0,Y)),

assert(visited\_state(X,Y)),

write("Empty the 4-Gallon jug on ground: (", X,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

state(X,Y):- Y > 0,

not(visited\_state(X,0)),

assert(visited\_state(X,0)),

write("Empty the 3-Gallon jug on ground: (", X,",",Y,") --> (", X,",",0,")\n"),

state(X,0).

state(X,Y):- X + Y >= 4,

Y > 0,

NEW\_Y = Y - (4 - X),

not(visited\_state(4,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour water from 3-Gallon jug to 4-gallon until it is full: (", X,",",Y,") --> (", 4,",",NEW\_Y,")\n"),

state(4,NEW\_Y).

state(X,Y):- X + Y >=3,

X > 0,

NEW\_X = X - (3 - Y),

not(visited\_state(X,3)),

assert(visited\_state(X,Y)),

write("Pour water from 4-Gallon jug to 3-gallon until it is full: (", X,",",Y,") --> (", NEW\_X,",",3,")\n"),

state(NEW\_X,3).

state(X,Y):- X + Y <=4,

Y > 0,

NEW\_X = X + Y,

not(visited\_state(NEW\_X,0)),

assert(visited\_state(X,Y)),

write("Pour all the water from 3-Gallon jug to 4-gallon: (", X,",",Y,") --> (", NEW\_X,",",0,")\n"),

state(NEW\_X,0).

state(X,Y):- X+Y<=3,

X > 0,

NEW\_Y = X + Y,

not(visited\_state(0,NEW\_Y)),

assert(visited\_state(X,Y)),

write("Pour all the water from 4-Gallon jug to 3-gallon: (", X,",",Y,") --> (", 0,",",NEW\_Y,")\n"),

state(0,NEW\_Y).

state(0,2):- not(visited\_state(2,0)),

assert(visited\_state(0,2)),

write("Pour 2 gallons from 3-Gallon jug to 4-gallon: (", 0,",",2,") --> (", 2,",",0,")\n"),

state(2,0).

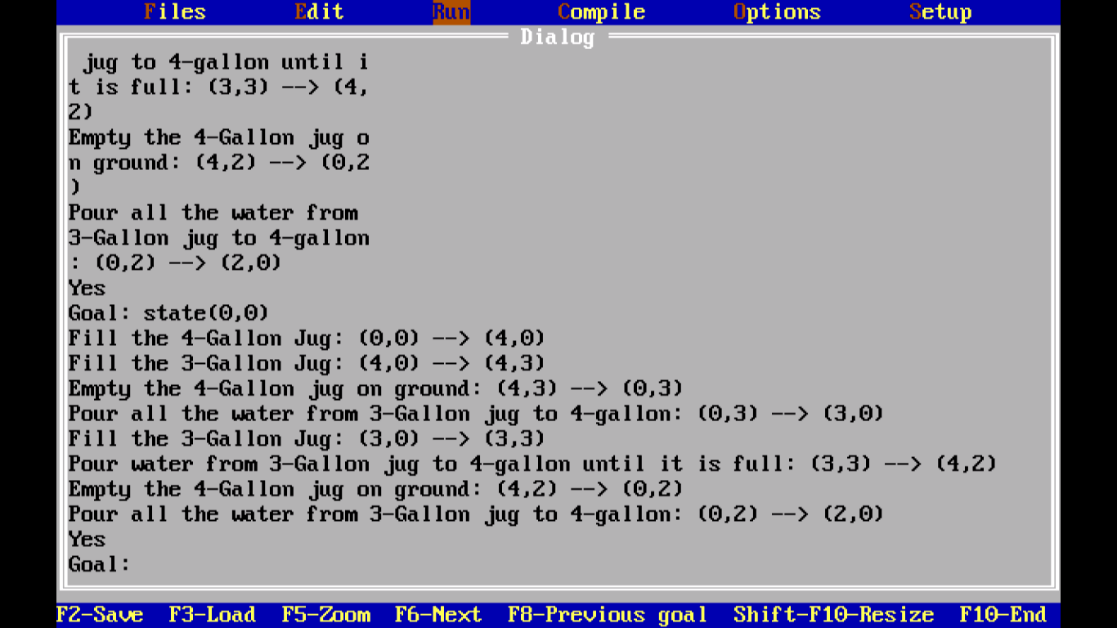
state(2,Y):- not(visited\_state(0,Y)),

assert(visited\_state(2,Y)),

write("Empty 2 gallons from 4-Gallon jug on the ground: (", 2,",",Y,") --> (", 0,",",Y,")\n"),

state(0,Y).

## Output: -



## Prolog program to solve the water-jug puzzle using DFS: -

domains

X,Y,Z=integer

predicates

state(integer,integer)

clauses

state(0,0):-write("Fill 3 litre jug"),nl,state(0,3).

state(0,3):-write("Pour everything from 3 to 4"),nl,state(3,0).

state(3,0):-write("Fill 3 litre jug"),nl,state(3,3).

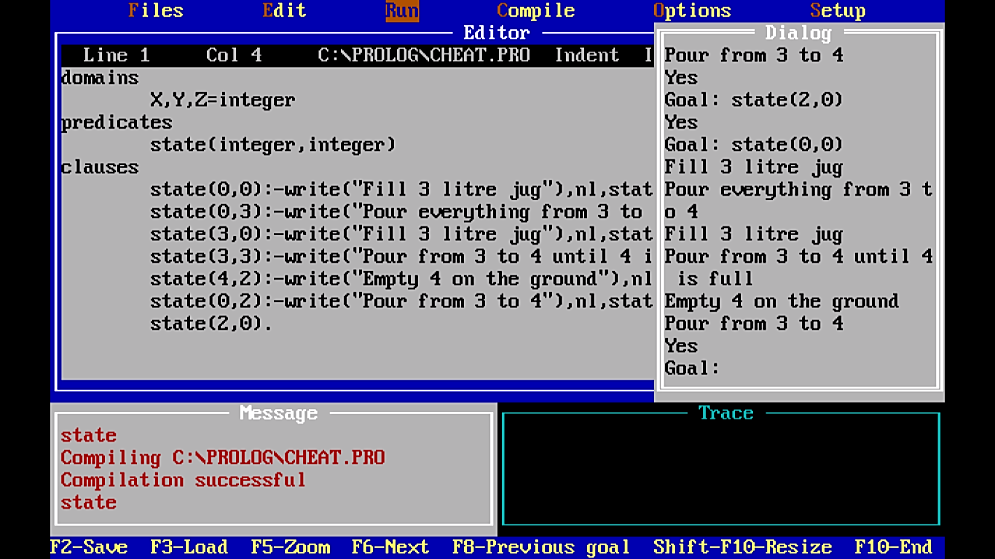
state(3,3):-write("Pour from 3 to 4 until 4 is full"),nl,state(4,2).

state(4,2):-write("Empty 4 on the ground"),nl,state(0,2).

state(0,2):-write("Pour from 3 to 4"),nl,state(2,0).

state(2,0).

## Output: -



# Practical – 10

# Aim: Write a program to implement Single Player Game (Using Heuristic Function).

#include <stdio.h>

#include <stdlib.h>

char matrix[3][3]; char check(void);

void init\_matrix(void);

void get\_player\_move(void);

void get\_computer\_move(void);

void disp\_matrix(void);

int main(void)

{

char done;

printf("This is the game of Tic Tac Toe.\n");

printf("You will be playing against the computer.\n");

done = ' ';

init\_matrix();

do {

disp\_matrix();

get\_player\_move();

done = check(); /\* see if winner \*/

if(done!= ' ') break; /\* winner!\*/

get\_computer\_move();

done = check(); /\* see if winner \*/

} while(done== ' ');

if(done=='X') printf("You won!\n");

else printf("I won!!!!\n");

disp\_matrix(); /\* show final positions \*/

return 0;

}

/\* Initialize the matrix. \*/

void init\_matrix(void)

{

int i, j;

for(i=0; i<3; i++)

for(j=0; j<3; j++) matrix[i][j] = ' ';

}

/\* Get a player's move. \*/

void get\_player\_move(void)

{

int x, y;

printf("Enter X,Y coordinates for your move: "); scanf("%d%\*c%d", &x, &y); x--; y--;

if(matrix[x][y]!= ' '){

printf("Invalid move, try again.\n");

get\_player\_move();

}

else matrix[x][y] = 'X';

}

/\* Get a move from the computer. \*/

void get\_computer\_move(void)

{

int i, j;

for(i=0; i<3; i++){

for(j=0; j<3; j++)

if(matrix[i][j]==' ') break;

if(matrix[i][j]==' ') break;

}

if(i\*j==9) {

printf("draw\n");

exit(0);

}

else

matrix[i][j] = 'O';

}

/\* Display the matrix on the screen. \*/

void disp\_matrix(void)

{

int t;

for(t=0; t<3; t++) {

printf(" %c | %c | %c ",matrix[t][0],

matrix[t][1], matrix [t][2]);

if(t!=2) printf("\n---|---|---\n");

}

printf("\n");

}

/\* See if there is a winner. \*/

char check(void)

{

int i;

for(i=0; i<3; i++) /\* check rows \*/

if(matrix[i][0]==matrix[i][1] &&

matrix[i][0]==matrix[i][2]) return matrix[i][0];

for(i=0; i<3; i++) /\* check columns \*/

if(matrix[0][i]==matrix[1][i] &&

matrix[0][i]==matrix[2][i]) return matrix[0][i];

/\* test diagonals \*/

if(matrix[0][0]==matrix[1][1] &&

matrix[1][1]==matrix[2][2])

return matrix[0][0];

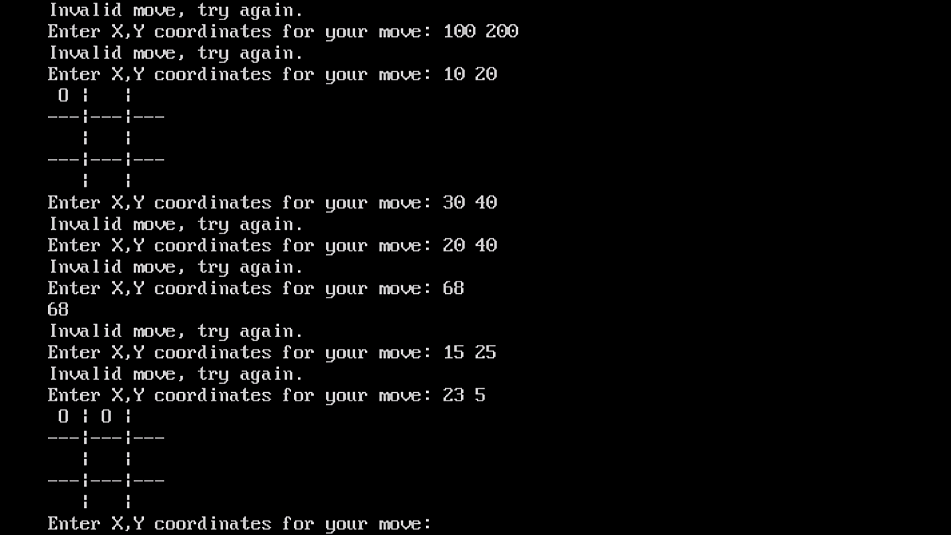
if(matrix[0][2]==matrix[1][1] && matrix[1][1]==matrix[2][0])

return matrix[0][2];

return ' ';

}

## Output: -



# Practical – 11

# Aim: Write a program to implement A\* algorithm.

## Python program to implement A\* algorithm: -

from \_\_future\_\_ import print\_function

import matplotlib.pyplot as plt

class AStarGraph(object):

# Define a class board like grid with two barriers

def \_\_init\_\_(self):

self.barriers = []

self.barriers.append([(2, 4), (2, 5), (2, 6), (3, 6), (4, 6),

(5, 6), (5, 5), (5, 4), (5, 3), (5, 2), (4, 2), (3, 2)])

def heuristic(self, start, goal):

# Use Chebyshev distance heuristic if we can move one square either

#adjacent or diagonal

D = 1

D2 = 1

dx = abs(start[0] - goal[0])

dy = abs(start[1] - goal[1])

return D \* (dx + dy) + (D2 - 2 \* D) \* min(dx, dy)

def get\_vertex\_neighbours(self, pos):

n = []

# Moves allow link a chess king

for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1), (1, 1), (-1, 1), (1, -1), (-1, -1)]:

x2 = pos[0] + dx

y2 = pos[1] + dy

if x2 < 0 or x2 > 7 or y2 < 0 or y2 > 7:

continue

n.append((x2, y2))

return n

def move\_cost(self, a, b):

for barrier in self.barriers:

if b in barrier:

return 100 # Extremely high cost to enter barrier squares

return 1 # Normal movement cost

def AStarSearch(start, end, graph):

G = {} # Actual movement cost to each position from the start position

F = {} # Estimated movement cost of start to end going via this position

# Initialize starting values

G[start] = 0

F[start] = graph.heuristic(start, end)

closedVertices = set()

openVertices = set([start])

cameFrom = {}

while len(openVertices) > 0:

# Get the vertex in the open list with the lowest F score

current = None

currentFscore = None

for pos in openVertices:

if current is None or F[pos] < currentFscore:

currentFscore = F[pos]

current = pos

# Check if we have reached the goal

if current == end:

# Retrace our route backward

path = [current]

while current in cameFrom:

current = cameFrom[current]

path.append(current)

path.reverse()

return path, F[end] # Done!

# Mark the current vertex as closed

openVertices.remove(current)

closedVertices.add(current)

# Update scores for vertices near the current position

for neighbour in graph.get\_vertex\_neighbours(current):

if neighbour in closedVertices:

continue # We have already processed this node exhaustively

candidateG = G[current] + graph.move\_cost(current, neighbour)

if neighbour not in openVertices:

openVertices.add(neighbour) # Discovered a new vertex

elif candidateG >= G[neighbour]:

continue # This G score is worse than previously found

# Adopt this G score

cameFrom[neighbour] = current

G[neighbour] = candidateG

H = graph.heuristic(neighbour, end)

F[neighbour] = G[neighbour] + H

raise RuntimeError("A\* failed to find a solution")

if \_\_name\_\_ == "\_\_main\_\_":

graph = AStarGraph()

result, cost = AStarSearch((0, 0), (7, 7), graph)

print("route", result)

print("cost", cost)

plt.plot([v[0] for v in result], [v[1] for v in result])

for barrier in graph.barriers:

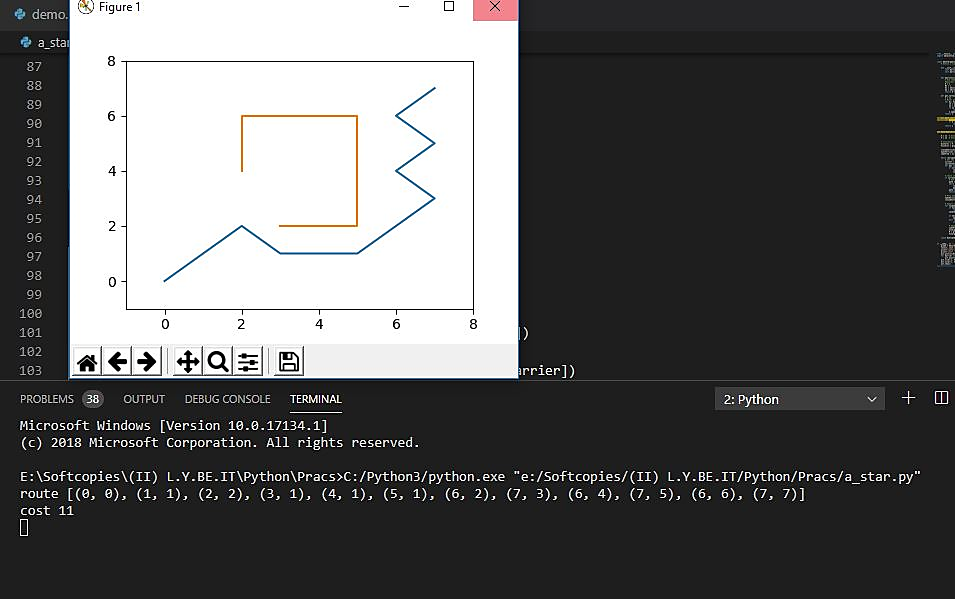
plt.plot([v[0] for v in barrier], [v[1] for v in barrier])

plt.xlim(-1, 8)

plt.ylim(-1, 8)

plt.show()

## Output: -



# Practical – 12

# Aim: N-Queens problem.

## Program for N-queens problem: -

n\_queens(N,Q) :-

length(Q,N),

board(Q,Board,0,N, \_, \_),

queens(Board,0,Q).

board([], [], N, N, \_, \_).

board([\_|Queens],

[Col-Vars|Board],

Col0, N, [\_|VR], VC) :-

Col is Col0+1,

functor(Vars, f, N),

constraints(N, Vars, VR, VC),

board(Queens, Board, Col, N, VR, [\_|VC]).

constraints(0, \_, \_, \_) :- !.

constraints(N, Row, [R|Rs], [C|Cs]) :-

arg(N, Row, R-C),

M is N-1,

constraints(M, Row, Rs, Cs).

queens([], \_, []).

queens([C|Cs], Row0, [Col|Solution]) :-

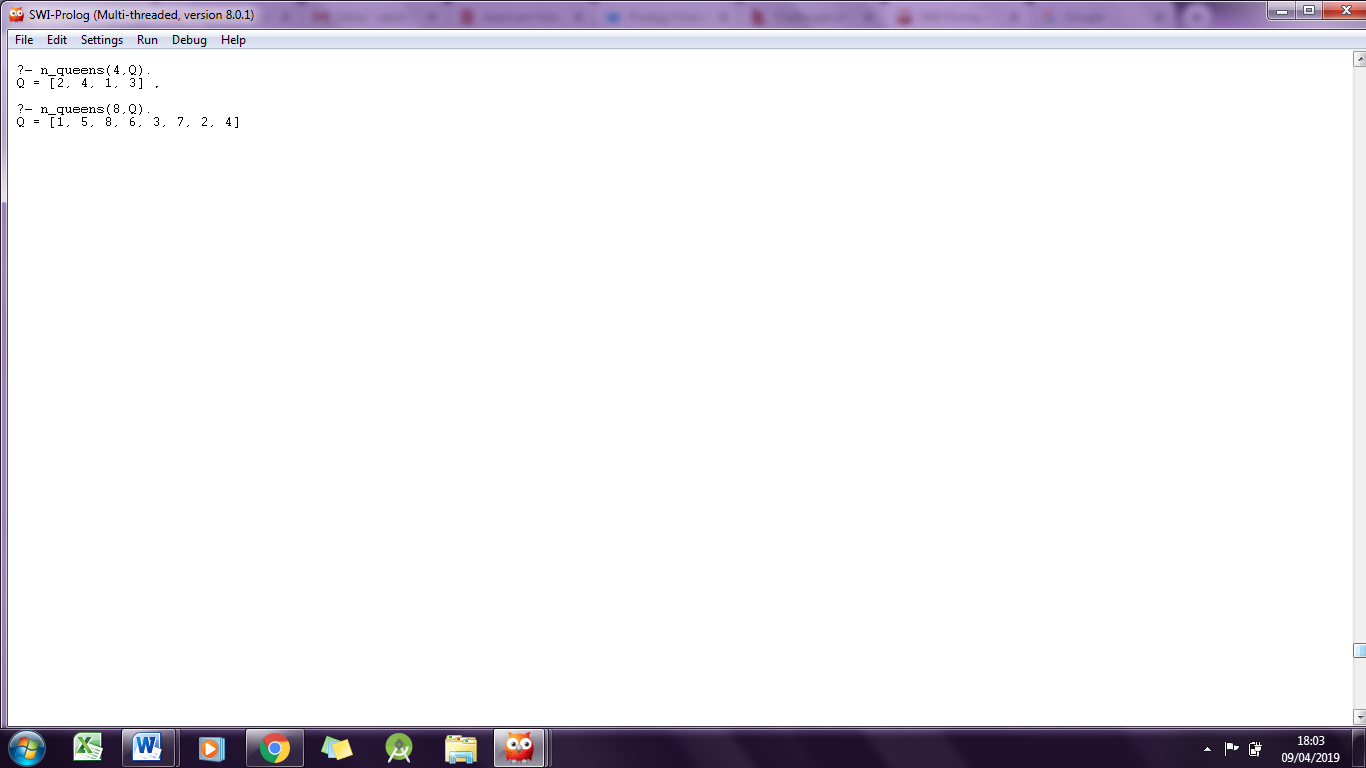
Row is Row0+1,

select(Col-Vars, [C|Cs], Board),

arg(Row, Vars, Row-Row),

queens(Board, Row, Solution).

Output: -



# Practical – 13

# Aim: 8-puzzle problem.

## Program to solve 8-puzzle problem: -

goal(1/2/3/8/0/4/7/6/5).

left( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J ).

left( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J ).

left( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J ).

left( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J ).

left( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J ).

left( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I ).

up( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J ).

up( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J ).

up( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J ).

up( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J ).

up( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J ).

up( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F ).

right( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J ).

right( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J ).

right( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 ).

right( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J ).

right( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J ).

right( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J ).

down( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J ).

down( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J ).

down( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 ).

down( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J ).

down( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J ).

down( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J ).

h\_function(Puzz,H) :- p\_fcn(Puzz,P),

s\_fcn(Puzz,S),

H is P + 3\*S.

move(P,C,left) :- left(P,C).

move(P,C,up) :- up(P,C).

move(P,C,right) :- right(P,C).

move(P,C,down) :- down(P,C).

%%% Manhattan distance

p\_fcn(A/B/C/D/E/F/G/H/I, P) :-

a(A,Pa), b(B,Pb), c(C,Pc),

d(D,Pd), e(E,Pe), f(F,Pf),

g(G,Pg), h(H,Ph), i(I,Pi),

P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.

a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).

b(0,0). b(1,0). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).

c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).

d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).

e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).

f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).

g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).

h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).

i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).

%%% the out-of-cycle function

s\_fcn(A/B/C/D/E/F/G/H/I, S) :-

s\_aux(A,B,S1), s\_aux(B,C,S2), s\_aux(C,F,S3),

s\_aux(F,I,S4), s\_aux(I,H,S5), s\_aux(H,G,S6),

s\_aux(G,D,S7), s\_aux(D,A,S8), s\_aux(E,S9),

S is S1+S2+S3+S4+S5+S6+S7+S8+S9.

s\_aux(0,0) :- !.

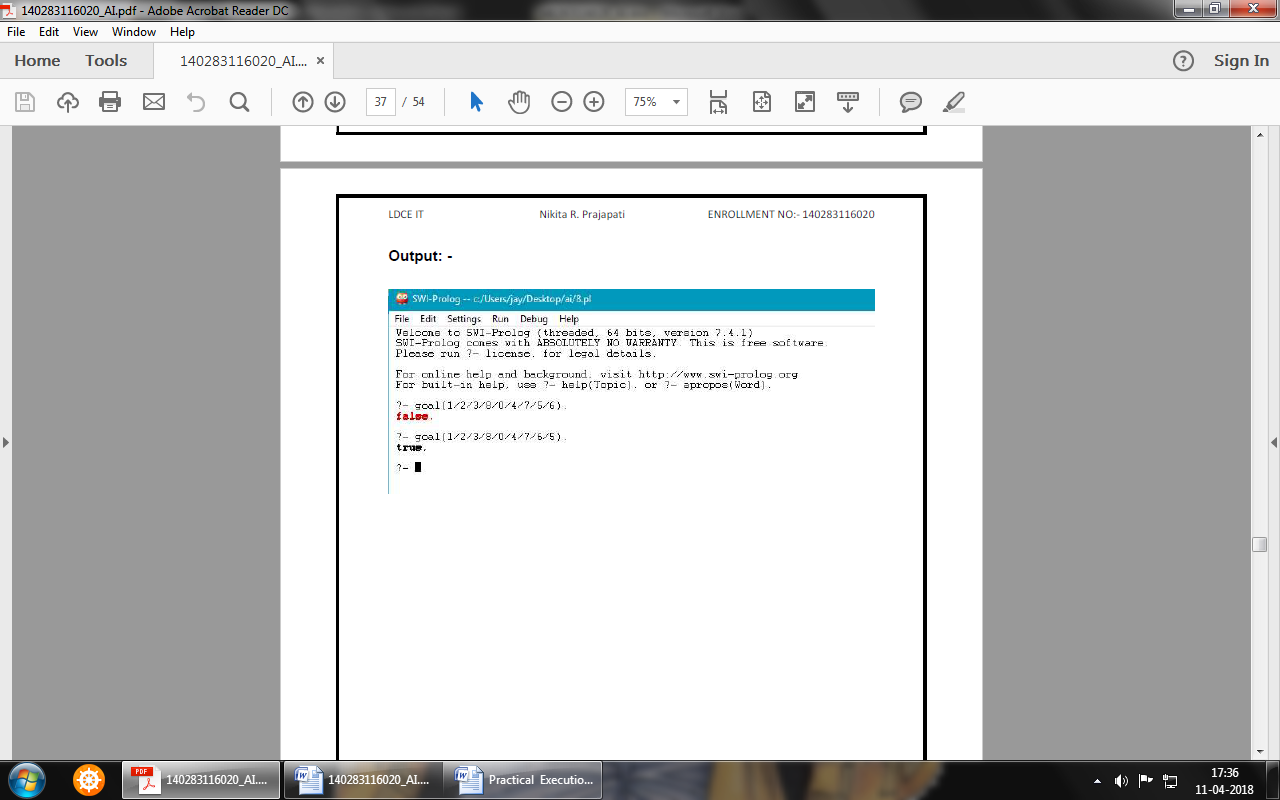
s\_aux(\_,1).

s\_aux(X,Y,0) :- Y is X+1, !.

s\_aux(8,1,0) :- !.

s\_aux(\_,\_,2).

## Output:



# Practical – 14

# Aim: Travelling salesman problem.

## Program to solve travelling salesman problem: -

data(1,[1,2,3],[1-1-2,2-5-1,2-2-3,3-1-2,1-4-3,3-3-1]).

data(2,[1,2,3,4,5],[1-4-2,2-1-1,1-4-3,3-1-1,2-5-3,3-2-2,1-2-4,4-2-1,2-5-4,4-4-2,3-4-4,4-1-3,1-1-5,5-1-1,2-3-5,5-2-2,3-0-5,5-1-3,4-9-5,5-8-4]).

tsp(V,E,Route,Sum):-

setof(ARoute-ASum,route(V,E,ARoute,ASum),Routes), %compute

all routes

min(Routes,Route-Sum). %find minimal route

min([H|T],X):-

min0(T,H,X). %H is minimum of List without tail T.

min0([],H,H). %actual minimum is overall minimum

min0([H-Hs|T],\_-As,X):- %actual minimum is greater than new head:

As >= Hs,

min0(T,H-Hs,X). %new head becomes actual minimum

min0([\_-Hs|T],A-As,X):- %other case

As =< Hs,

min0(T,A-As,X). %keep actual minimum

route(V,E,Route,Sum):-

select(V,1,V1), %remove start node

visit\_all(1,V1,E,[1],Route0,0,Sum0), %visit all other nodes

Route0=[H|\_],

member(H-D-1,E), %return to node 1

Route=[1|Route0],

Sum is D+Sum0.

visit\_all(\_,[],\_,R,R,Sum,Sum). %base case: no vertices left

visit\_all(N,V,E,Rin,Rout,Sumin,Sumout):-

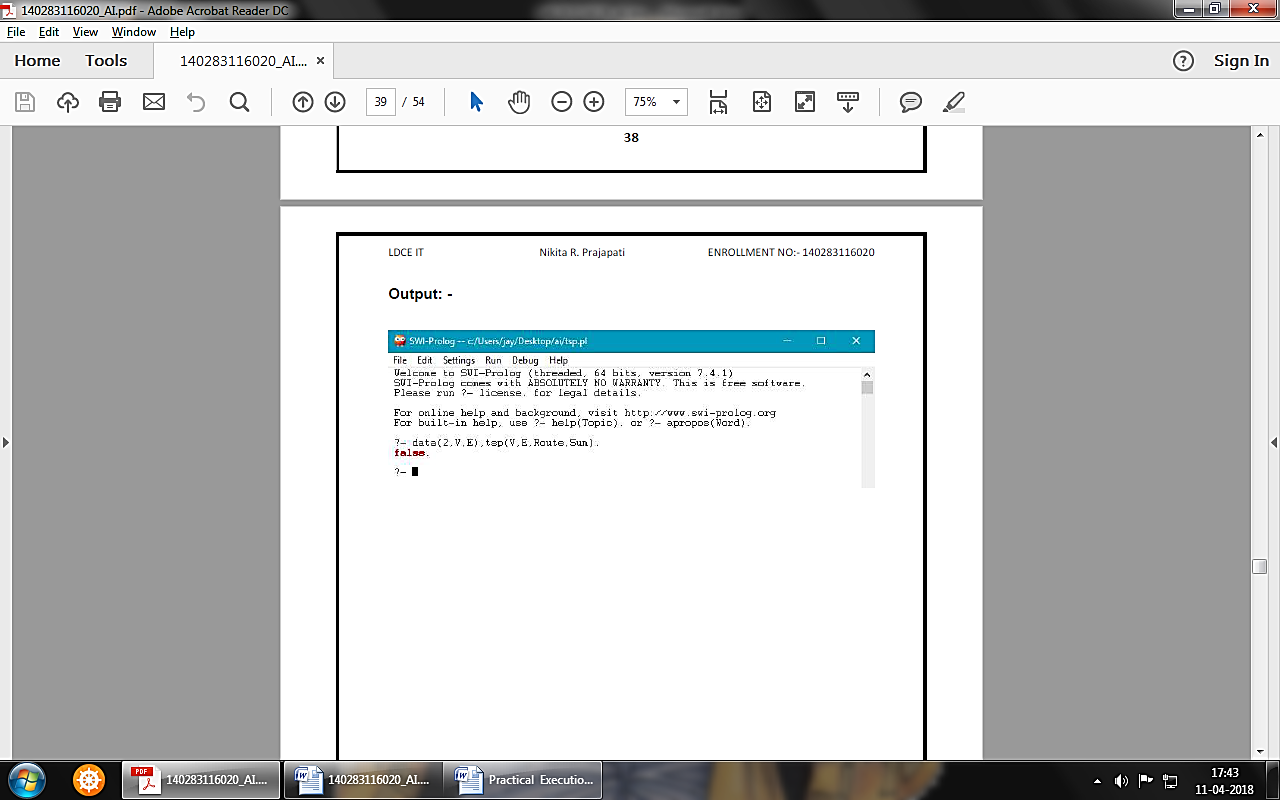
select(V,N1,V1), %select a vertice

member(N-D-N1,E), %find edge (get distance)

Sumout0 is Sumin+D, %update distance-sum

visit\_all(N1,V1,E,[N1|Rin],Rout,Sumout0,Sumout). %keep going

## Output:-



# Practical – 15

# Aim: Convert following Prolog predicates into Semantic Net.

cat(tom).

cat(cat1).

mat(mat1).

sat\_on(cat1,mat1).

bird(bird1).

caught(tom,bird1).

like(X,cream) :– cat(X).

mammal(X) :– cat(X).

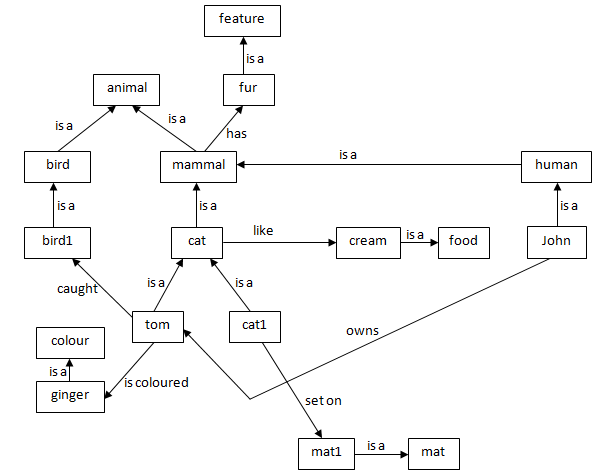
has(X,fur) :– mammal(X).

animal(X) :– mammal(X).

animal(X) :– bird(X).

owns(john,tom).

is\_coloured(tom,ginger).



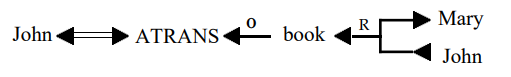
# Practical – 16

# Aim: Write the Conceptual Dependency for following statements.

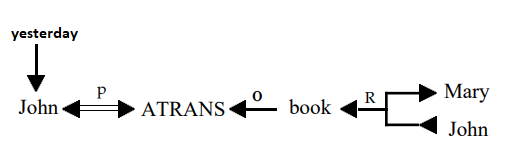
*(a)* John gives Mary a book.

*(b)* John gave Mary the book yesterday.

*(a)* John gives Mary a book.



*(b)* John gave Mary the book yesterday.



Beyond Syllabus Practical

# Aim: Write a PROLOG program to solve tower of Hanoi problem.

## Program:

domains

loc =right;middle;left

predicates

hanoi(integer)

move(integer,loc,loc,loc)

inform(loc,loc)

clauses

hanoi(N):- move(N,left,middle,right).

move(1,A,\_,C):- inform(A,C),!.

move(N,A,B,C):-

N1=N-1, move(N1,A,C,B),

inform(A,C),move(N1,B,A,C).

inform(Loc1, Loc2):-nl,

write("Move a disk from ", Loc1, " to ", Loc2).

## Output:

