1. Writesimplefactforfollowing:

* 1. Ramlikesmango.
  2. Seemaisagirl.
  3. BilllikesCindy.
  4. Roseisred.
  5. Johnownsgold

Solution:

% Facts

1. Ram likes mango.

2. Seema is a girl.

3. Bill likes Cindy.

4. Rose is red.

5. John owns gold.

% Clauses

likes(ram ,mango).

girl(seema).

red(rose).

likes(bill ,cindy).

owns(john ,gold).

Source code:

likes(ram ,mango).

girl(seema).

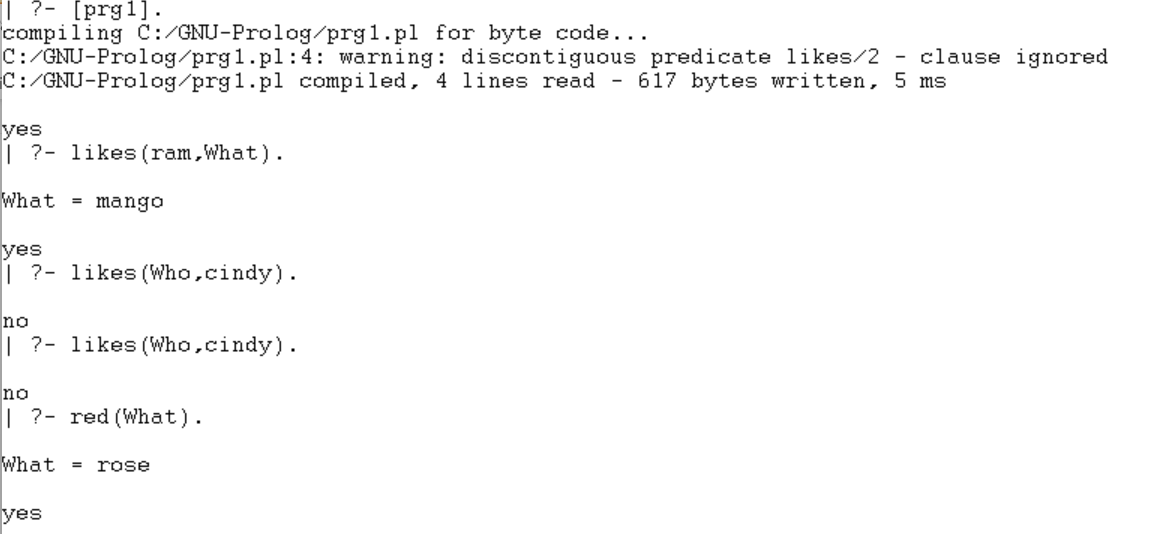
red(rose).

likes(bill ,cindy).

owns(john ,gold).

Execution:[prg1].

Output:



1. Write predicates one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

Solution:

% Production rules:

c\_to\_f

f is c \* 9 / 5 + 32

freezing f < = 32

% Rules:

c\_to\_f(C,F) :-

F is C \* 9 / 5 + 32.

freezing(F) :-

F =< 32.

Source Code:

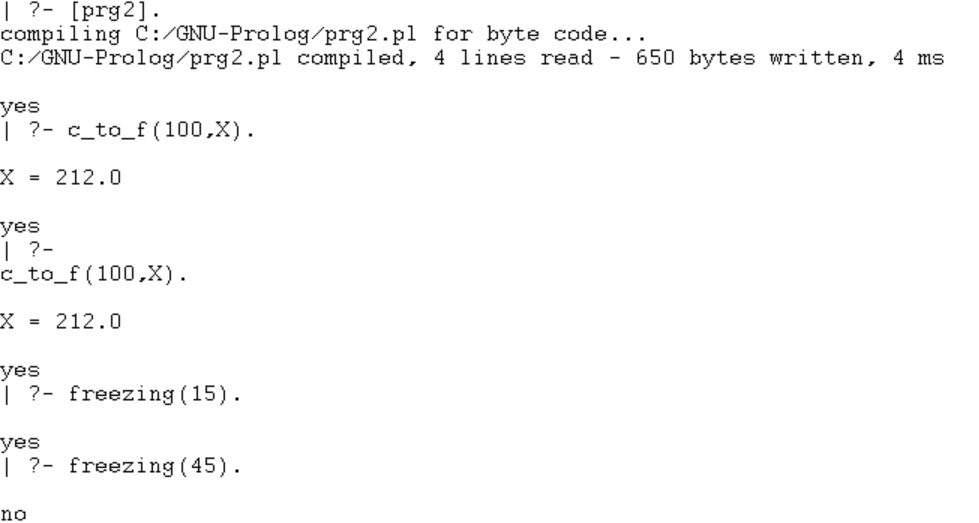
c\_to\_f(C,F) :-

F is C \* 9 / 5 + 32.

freezing(F) :-

F =< 32.

Execution:[prg2].

Output: 

1. Write a program to solve the Monkey Banana problem

/\* Description: Imagine a room containing a monkey, chair and some bananas. That have been hanged from the center of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair . The problem is to prove the monkey can reach the bananas. The monkey can perform the following actions: 1) Walk on the floor 2) Climb the box 3) Push the box around(if it is beside the box). 4) Grasp the banana if it is standing on the box directly under the banana. \*/

% Production rules:

can\_reach -> clever,close.

get\_on: -> can\_climb.

under -> in room,in\_room, in\_room,can\_climb.

Close -> get\_on,under | tall

% Clauses:

in\_room(bananas).

in\_room(chair).

in\_room(monkey).

clever(monkey).

can\_climb(monkey, chair).

tall(chair).

can\_move(monkey, chair, bananas).

can\_reach(X, Y):-clever(X),close(X, Y).

get\_on(X,Y):-

can\_climb(X,Y).

under(Y,Z):-

in\_room(X),in\_room(Y),

in\_room(Z),can\_climb(X,Y,Z).

close(X,Z):-

get\_on(X,Y), under(Y,Z);

tall(Y).

% Queries:

?- can\_reach(A, B).

A = monkey.

B = banana.

?- can\_reach(monkey, banana).

Yes.

Experiment 4:

Aim:

WAP in turbo prolog for medical diagnosis and show the advantages and disadvantages of green and red cuts

/\* Description:

This object of this famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for N=3 disks.

\*/

% Production rules:

hanoi(N) 🡪 move(N,left,middle,right).

move(1,A,\_,C) 🡪 inform(A,C),fail.

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

% 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):-

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

Source Code:

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):-

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

Output:

% Queries:

?- can\_reach(A, B).

A = monkey.

B = banana.

?- can\_reach(monkey, banana).

Yes.

Experiment 5:

Aim: Write a program to solve 4-Queens problem

Solution:

/\* Description:

In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

\*/

% Domains:

queen = q(integer, integer)

queens = queen\*

freelist = integer\*

board = board(queens, freelist, freelist, freelist, freelist)

% Predicates:

nondeterm placeN(integer, board, board)

nondeterm place\_a\_queen(integer, board, board)

nondeterm nqueens(integer)

nondeterm makelist(integer, freelist)

nondeterm findandremove(integer, freelist, freelist)

nextrow(integer, freelist, freelist)

% Clauses

nqueens(N):-

makelist(N,L),

Diagonal=N\*2-1,

makelist(Diagonal,LL),

placeN(N,board([],L,L,LL,LL),Final),

write(Final).

placeN(\_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.

placeN(N,Board1,Result):-

place\_a\_queen(N,Board1,Board2),

placeN(N,Board2,Result).

place\_a\_queen(N,

board(Queens,Rows,Columns,Diag1,Diag2),

board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-

nextrow(R,Rows,NewR),

findandremove(C,Columns,NewC),

D1=N+C-R,findandremove(D1,Diag1,NewD1),

D2=R+C-1,findandremove(D2,Diag2,NewD2).

findandremove(X,[X|Rest],Rest).

findandremove(X,[Y|Rest],[Y|Tail]):-

findandremove(X,Rest,Tail).

makelist(1,[1]).

makelist(N,[N|Rest]) :-

N1=N-1,makelist(N1,Rest).

nextrow(Row,[Row|Rest],Rest).

Source Code:

nqueens(N):-

makelist(N,L),

Diagonal=N\*2-1,

makelist(Diagonal,LL),

placeN(N,board([],L,L,LL,LL),Final),

write(Final).

placeN(\_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.

placeN(N,Board1,Result):-

place\_a\_queen(N,Board1,Board2),

placeN(N,Board2,Result).

place\_a\_queen(N,

board(Queens,Rows,Columns,Diag1,Diag2),

board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-

nextrow(R,Rows,NewR),

findandremove(C,Columns,NewC),

D1=N+C-R,findandremove(D1,Diag1,NewD1),

D2=R+C-1,findandremove(D2,Diag2,NewD2).

findandremove(X,[X|Rest],Rest).

findandremove(X,[Y|Rest],[Y|Tail]):-

findandremove(X,Rest,Tail).

makelist(1,[1]).

makelist(N,[N|Rest]) :-

N1=N-1,makelist(N1,Rest).

nextrow(Row,[Row|Rest],Rest).

Output:

% Goal

nqueens(4),nl.

board([q(1,2),q(2,4),q(3,1),q(4,3),[],[],[7,4,1],[7,4,1])

yes

Experiment-6

Aim: Write a program to solve Traveling salesman problems

Solution:

/\* Description:

For example, there are four cities(Kansas City,Houston,Gordon and Tampa).

-> The distance between Kansas City and Houston is 120.

-> The distance between Kansas City and Tampa is 80.

-> The distance between Houston and Gordon is 100.

\*/

% Production Rules:-

route(Town1,Town2,Distance)🡪 road(Town1,Town2,Distance).

route(Town1,Town2,Distance)🡪 road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

% Domains

town = symbol

distance = integer

% Predicates

nondeterm road(town,town,distance)

nondeterm route(town,town,distance)

% Clauses

road("tampa","houston",200).

road("gordon","tampa",300).

road("houston","gordon",100).

road("houston","kansas\_city",120).

road("gordon","kansas\_city",130).

route(Town1,Town2,Distance):-

road(Town1,Town2,Distance).

route(Town1,Town2,Distance):-

road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

!.

Source Code:

road("tampa","houston",200).

road("gordon","tampa",300).

road("houston","gordon",100).

road("houston","kansas\_city",120).

road("gordon","kansas\_city",130).

route(Town1,Town2,Distance):-

road(Town1,Town2,Distance).

route(Town1,Town2,Distance):-

road(Town1,X,Dist1),

route(X,Town2,Dist2),

Distance=Dist1+Dist2,

!.

**Output:**

% Goal

route("tampa", "kansas\_city", X),

write("Distance from Tampa to Kansas City is ",X),nl.

Distance from Tampa to Kansas City is 320

X=320

1 Solution