Turbo Prolog Editor

Practical No: 1 **Date:**

Aim: To study the turbo prolog editor and implement basic facts.

Exercise:

1. Write a Prolog program that demonstrates the below given facts:

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

Ans. Clauses

a. likes(ram ,mango).
b. girl(seema).
c. red(rose).
d. likes(bill ,Cindy).
e.owns(John ,gold)

O/P: Goal

queries

?-likes(ram,What).

Wha t= mango

?-likes(Who, Cindy).
Who = Cindy

?-red(What).
What = rose

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

Study of family relationships Tree

Practical No: 2 **Date:**

Aim: To study about family relationship tree in Prolog.

Exercise:

1. Write a Prolog program for family relationship tree.

A. Program

```
female(pa).
female(liz).
female(pat).
female(ann).
male(Jim).
male(bob).
male(tom).
male(peter).
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
parent(bob,peter).
parent(peter,jim).
mother(X,Y):- parent(X,Y),female(X).
father(X,Y):- parent(X,Y),male(X).
```

```
haschild(X):- parent(X,_).
sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.
```

O/P:

```
yes | ?-
parent(X,jim).
X = pat ? ;

X = peter

yes | ?-
mother(X,YX = pam
Y = bob ? ;

X = pat
Y = jim ? ;

no
| ?- haschild(X).

X = pam ? ;

X = tom ? ;

X = tom ? ;

X = bob ? ;

X = bob ? ;
```

X = pat ? ;

X = bob ? ;

X = peter

yes
| ?- sister(X,Y).

X = liz
Y = bob ? ;

X = ann
Y = pat ? ;

X = ann
Y = peter ? ;

X = pat
Y = ann ? ;

X = pat
Y = peter ? ;

(16 ms) no | ?

Study of rules in prolog

Practical No: 3 **Date:**

Aim: To study rules implemented in Prolog.

Exercise:

1. Write a prolog program to create simple calculator using prolog.

Program

```
PREDICATES
ADD(integer,integer,integer)
SUB(integer,integer,integer)
MUL(integer,integer,integer)
DIV(integer,integer,integer)

CLAUSES
ADD(A,B,SUM):-
SUM=A+B.
SUB(A,B,DIF):-
DIF=A-B.
MUL(A,B,MUL):-
MUL=A*B.
DIV(A,B,DIV):-
DIV=A/B.

//output
Goal: ADD(5,4,SUM)
SUM=9
```

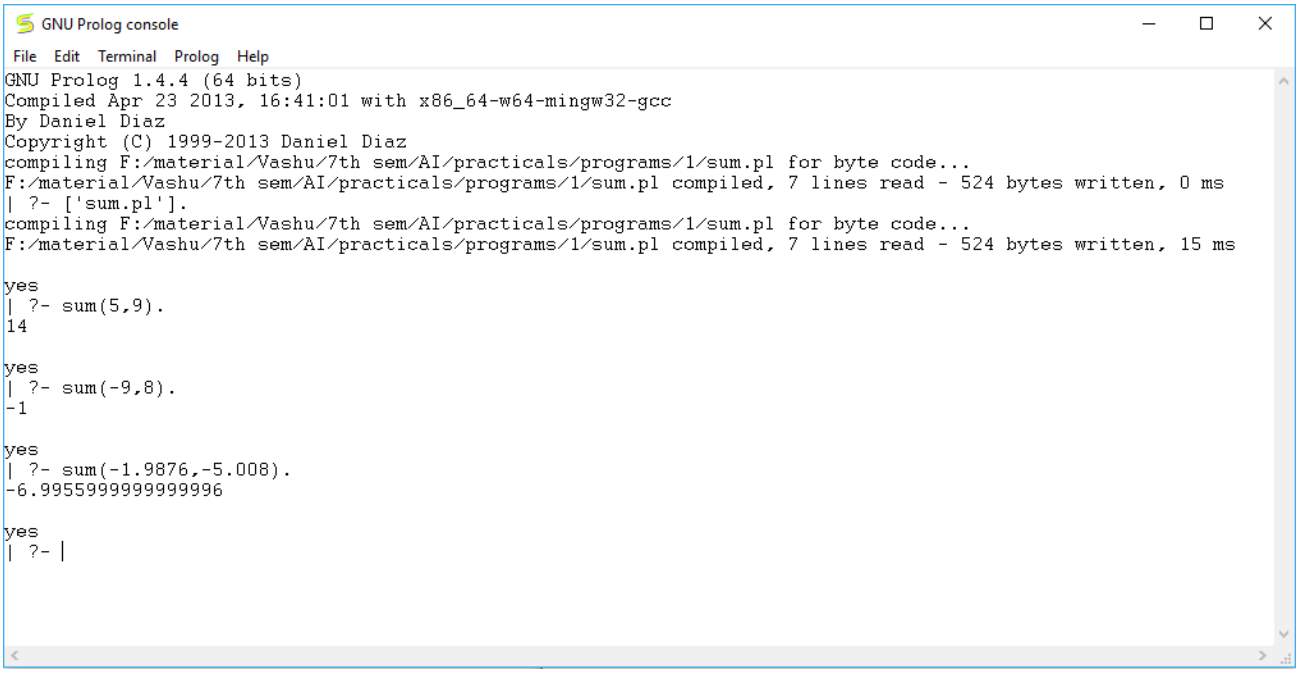
1 solution
Goal: SUB(9,4,SUB)
SUB=5
1 solution
Goal: MUL(4,5,MUL)
MUL=20
1 solution
Goal: DIV(10,2,DIV)
DIV=5
1 solution

2. Write a prolog program take as input 2 integers values and finds out the greater number

Program

```
sum(x,y):-
S is X+Y,
write(S)
```

Output:



Study of recursion in prolog

Practical No: 4 **Date:**

Aim: To study how to implement recursion in Prolog.

Exercise:

1. Write a prolog program to print factorial of a number.

predicates factorial(integer, real) go clauses go if
write("Enter a positive integer number:"), readint(N),
factorial(N,Result),
write("Factorial of", N, "is=", Result). factorial(0, 1)
factorial(N, Result) if N>0, N1=N-1, factorial(N1, Res),
Result=N*Res.

O/P

goals:
factorial(5, Answer) Answer=120

2. Write a prolog program to implement tower of hanoi.

Program:

```
move(1,X,Y,_) :-
 write('Move top disk from '), write(X),
write(' to '), write(Y), nl. move(N,X,Y,Z)
:-
 N>1,
```

```
 M is N-1,
 move(M,
 X,Z,Y),
 move(1,X
 ,Y,_),
 move(M,
 Z,Y,X).
```

O/P:

```
| ?- [towersofhanoi].
compiling D:/TP Prolog/Sample_Codes/towersofhanoi.pl for byte
code...
D:/TP Prolog/Sample_Codes/towersofhanoi.pl compiled,
8 lines read - 1409 bytes written, 15 ms

yes
| ?-
move(4,source,target,au
xiliary). Move top disk
from source to auxiliary
Move top disk from
source to target Move
top disk from auxiliary
to target Move top disk
from source to auxiliary
Move top disk from
target to source Move
top disk from target to
```

auxiliary Move top disk
from source to auxiliary
Move top disk from
source to target Move
top disk from auxiliary
to target Move top disk
from auxiliary to source
Move top disk from
target to source Move
top disk from auxiliary
to target Move top disk
from source to auxiliary
Move top disk from
source to target Move
top disk from auxiliary
to target

true ?

(31 ms) ye

Study of GCD in prolog

Practical No: 5 **Date:**

Aim: To study the use of GCD in Prolog.

Exercise:

1. Write a prolog program to find GCD number.

Program

```
predicates
gcd(integer, integer, integer)

clauses
gcd(M, O, M). gcd(M, N, Result):-
Rem=M mod N,
gcd(N, Rem, Result).
```

O/P:

```
goals:
gcd(6, 4, Result) Result=2
```

Study of lists in prolog

Practical No: 6 **Date:**

Aim: To study the use of List in Prolog.

Exercise:

1. Write a prolog program to print the contents of a list.

Program:

```
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).
```

O/P:

```
| ?- [list_basics].
compiling D:/TP Prolog/Sample_Codes/list_basics.pl for byte code...
D:/TP Prolog/Sample_Codes/list_basics.pl compiled, 1 lines read - 467 bytes written, 13 ms

yes
| ?- list_member(b,[a,b,c]).

true ?

yes
| ?- list_member(b,[a,[b,c]]).
```

```
no
| ?- list_member([b,c],[a,[b,c]]).

true ?

yes
| ?- list_member(d,[a,b,c]).

no
| ?- list_member(d,[a,b,c])
```

2. Write a prolog program to input integer numbers in the list

Program:

```
 | ?- write(56).
```

Output 5

3. Write a prolog program to find the sum of integer list.

Program:

```
  domains
  x = integer l = integer*
  predicates sum(l,x)
  clauses sum([],0).
  sum([X|List],Sum) :-sum(List,Sum1), Sum = X + Sum1
```

Study of water jug problem in prolog

Practical No: 7 **Date:**

Aim: To study how to implement water jug problem in Prolog using BFS.

Exercise:

1. Write a program to implement BFS for Water Jug Problem

```
#include <bits/stdc++.h>
using namespace std;
int x;
int y;
void show(int a, int b);
int min(int w, int z)
{
  if (w < z)
     return w;
  else
     return z;
}
void show(int a, int b)
{
  cout << setw(12) << a << setw(12) << b << endl;
}
void s(int n)
{
  int xq = 0, yq = 0;
```

```
    int t;
    cout << setw(15) << "FIRST JUG" << setw(15) << "SECOND
JUG" << endl;
    while (xq != n && yq != n)
    {
        if (xq == 0)
        {
            xq = x;
            show(xq, yq);
        }
        else if (yq == y)
        {
            yq = 0;
            show(xq, yq);
        }
        else
        {
            t = min(y - yq, xq);
            yq = yq + t;
            xq = xq - t

                        show(xq, yq);
        }
    }
  }

  int main()
  {
    int n;
    cout << "Enter the liters of water required out of the two jugs:
";
    cin >> n;
```

```
    cout << "Enter the capacity of the first jug: ";
    cin >> x;
    cout << "Enter the capacity of the second jug: ";
    cin >> y;

    if (n < x || n < y)
    {
        if (n % (__gcd(x, y)) == 0)
            s(n);
        else
            cout << "This is not possible....\n";
    }
    else
        cout << "This is not possible....\n";
}
```

O/P :

```
Enter the liters of water required out of the two jugs: 3
Enter the capacity of the first jug: 5
Enter the capacity of the second jug: 4
      FIRST JUG     SECOND JUG
          5             0
          1             4
          1             0
          0             1
          5             1
          2             4
          2             0
          0             2
          5             2
          3             4


...Program finished with exit code 0
Press ENTER to exit console.
```

Study of N-queens problem in prolog

Practical No: 8 **Date:**

Aim: To study how to implement N-Queens problem in Prolog.

Exercise:

1. Write a program to solve 8-Queens problem using Prolog.

Program:

```
use_rendering(chess).

%%    queens(+N, -Queens) is nondet.
%
% @param     Queens is a list of column numbers for placing the queens.
% @author Richard A. O'Keefe (The Craft of Prolog)

queens(N, Queens) :-
   length(Queens, N),
    board(Queens, Board, 0, N, _, _),
    queens(Board, 0, Queens).

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).
```

Study of 8 puzzle problem in prolog

Practical No: 9 **Date:**

Aim: To study how to implement 8 puzzle problem in Prolog.

Exercise:

1. Solve the program for the sequence (8,7,6,5,4,1,2,3,0)

Program:

```
% Simple Prolog Planner for the 8 Puzzle Problem


% This predicate initialises the problem states. The first argument
% of solve/3 is the initial state, the 2nd the goal state, and the
% third the plan that will be produced.


test(Plan):-
    write('Initial state:'),nl,
    Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7),
at(tile1,8), at(tile7,9)],
    write_sol(Init),
    Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7),
at(tile7,8), at(tile8,9)],
    nl,write('Goal state:'),nl,
    write(Goal),nl,nl,
    solve(Init,Goal,Plan).


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


%Determines whether Current and Destination tiles are a valid move.
is_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).



% This predicate produces the plan. Once the Goal list is a subset
```

```
% of the current State the plan is complete and it is written to
% the screen using write_sol/1.


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).


% The problem has three operators.
% 1st arg = name
% 2nd arg = preconditions
% 3rd arg = delete list
% 4th arg = add list.


% Tile can move to new position only if the destination tile is empty & Manhattan
distance = 1
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)]).



% Utility predicates.


% Check is first list is a subset of the second


is_subset([H|T], Set):-
    member(H, Set),
    is_subset(T, Set).
is_subset([], _).


% Remove all elements of 1st list from second to create third.


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).



%----------------------Output Queries--------------------------------->


?- test(Plan).
Initial state:
at(tile7,9)
at(tile1,8)
at(tile5,7)
at(tile6,6)
at(tile2,5)
at(empty,4)
at(tile8,3)
at(tile3,2)
at(tile4,1)


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


false.
```

Study of travelling salesman problem using Prolog

Practical No: 10 **Date:**

Aim: To study how to implement travelling salesman problem using Prolog.

Exercise:

1. Write a prolog program for solving travelling sales problem consisting of 4 cities

Program

```
domains
/* will allow us cooperate with better names, for me this is like #typedef in C++ */

town = symbol

distance = unsigned

rib = r(town,town,distance)

tlist = town*

rlist = rib*

predicates

nondeterm way(town,town,rlist,distance)

nondeterm route(town,town,rlist,tlist,distance)

nondeterm route1(town,tlist,rlist,tlist,distance)
```

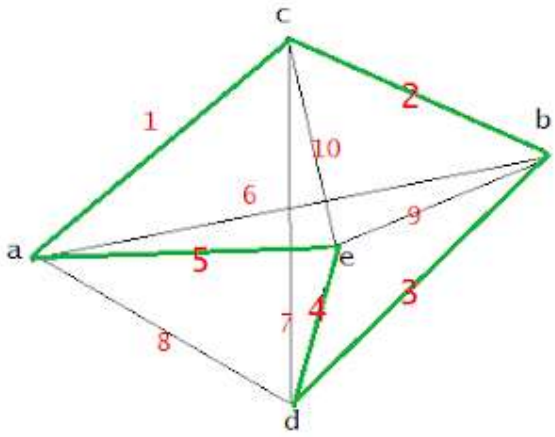
```
nondeterm ribsmember(rib,rlist)

nondeterm townsmember(town,tlist)

nondeterm tsp(town,town,tlist,rlist,tlist,distance)

nondeterm ham(town,town,tlist,rlist,tlist,distance)

nondeterm shorterRouteExists(town,town,tlist,rlist,distance)

nondeterm alltown(tlist,tlist)

nondeterm write_list(tlist)

clauses

/*

Nothing special with write_list.

If list is empty we do nothing,

and if something there we write head and call ourselves for tail.

*/

write_list([]).

write_list([H|T]):-

write(H,‘ ‘),

write_list(T).

/* Is true if town X is in list of towns… */

townsmember(X,[X|_]).

townsmember(X,[_|L]):-

townsmember(X,L).
```

```
/* Is true if rib X is in list of ribs… */

ribsmember(r(X,Y,D),[r(X,Y,D)|_]).

ribsmember(X,[_|L]):-

ribsmember(X,L).

/* Is true if Route consists of all Towns presented in second argument */

alltown(_,[]).

alltown(Route,[H|T]):-

townsmember(H,Route),

alltown(Route,T).


/* Is true if there is a way from Town1 to Town2, and also return distance between them */

way(Town1,Town2,Ways,OutWayDistance):-

ribsmember(r(Town1,Town2,D),Ways),

OutWayDistance = D.


%/*

/* If next is uncommented then we are using non-oriented graph*/

way(Town1,Town2,Ways,OutWayDistance):-

ribsmember(r(Town2,Town1,D),Ways), /*switching direction here…*/

OutWayDistance = D.
```

```
%*/

/* Is true if we could build route from Town1 to Town2 */

route(Town1,Town2,Ways,OutRoute,OutDistance):-

route1(Town1,[Town2],Ways,OutRoute,T1T2Distance),

%SWITCH HERE

way(Town2,Town1,Ways,LasDist), /* If you want find shortest way comment this line*/

OutDistance = T1T2Distance + LasDist. /* And make this: OutDistance = T1T2Distance.*/



route1(Town1,[Town1|Route1],_,[Town1|Route1],OutDistance):-

OutDistance = 0.

/* Does the actual finding of route. We take new TownX town and if it is not member of PassedRoute,

we continue searching with including TownX in the list of passed towns.*/

route1(Town1,[Town2|PassedRoute],Ways,OutRoute,OutDistance):-

way(TownX,Town2,Ways,WayDistance),

not(townsmember(TownX,PassedRoute)),

route1(Town1,[TownX,Town2|
PassedRoute],Ways,OutRoute,CompletingRoadDistance),

OutDistance = CompletingRoadDistance + WayDistance.



shorterRouteExists(Town1,Town2,Towns,Ways,Distance):-
```

```
    ham(Town1,Town2,Towns,Ways,_,Other),

    Other < Distance.

    /* calling tsp(a,a,…. picks any one connected to a town and calls another tsp*/

    tsp(Town1,Town1,Towns,Ways,BestRoute,MinDistance):-

    way(OtherTown,Town1,Ways,_),

    tsp(Town1,OtherTown,Towns,Ways,BestRoute,MinDistance).

    /*Travelling Salesman Problem is Hammilton way which is the shortes of other
ones.*/

    tsp(Town1,Town2,Towns,Ways,BestRoute,MinDistance):-

    ham(Town1,Town2,Towns,Ways,Route,Distance),

    not(shorterRouteExists(Town1,Town2,Towns,Ways,Distance))
    BestRoute = Route,

    MinDistance = Distance.

    /*Hammilton route from Town1 to Town2 assuming that Town2->Town1 way
exists.*/

    ham(Town1,Town2,Towns,Ways,Route,Distance):-

    route(Town1,Town2,Ways,Route,Distance),

    %SWITCH HERE

    alltown(Route,Towns), % if you want simple road without including all towns you
could uncomment this line

    write_list(Route),

    write(" tD = ",Distance,"n").

    % fail.
```

```
goal

/* EXAMPLE 1

AllTowns = [a,b,c,d],

AllWays = [r(a,b,1),r(a,c,10),r(c,b,2),r(b,c,2),r(b,d,5),r(c,d,3),r(d,a,4)],

*/

/* EXAMPLE 2 */

AllTowns = [a,b,c,d,e],

AllWays =
[r(a,c,1),r(a,b,6),r(a,e,5),r(a,d,8),r(c,b,2),r(c,d,7),r(c,e,10),r(b,d,3),r(b,e,9),r(d,e,4)] ,

tsp(a,a,AllTowns,AllWays,Route,Distance),

%SWITCH HERE

% tsp(a,b,AllTowns,AllWays,Route,Distance),

write(“Finally:n“),

write_list(Route),

write(” tMIN_D = “,Distance,“n“).
```



O/P:

```
a e d b c    D = 15 a
e d b c      D = 15 a
d e b c      D = 24 a
e b d c      D = 25 a
b e d c      D = 27 a
d b e c      D = 31 a
b d e c      D = 24
```

Finally:

```
a e d b c    MIN_D = 15
```