Fr. Conceicao Rodrigues College of Engineering
Fr. Agnel Ashram, Bandstand, Bandra (W), Mumbai - 400050

Department of Computer Engineering Academic Term II: 23-24

Class: B.E (Computer), Sem – VI **Subject Name: Artificial Intelligence**

Student Name: Siddhesh Pradhan **Roll No: 9632**

Practical No:	**5**
Title:	Programming in PROLOG
Date of Performance:	
Date of Submission:	

Rubrics for Evaluation:

Sr. No	**Performance Indicator**	**Excellent**	**Good**	**Below Average**	**Marks**
1	On time Completion & Submission (01)	01 (On Time)	NA	00 (Not on Time)	
2	Logic/Algorithm Complexity analysis (03)	03(Correct)	02(Partial)	01 (Tried)	
3	Coding Standards (03): Comments/indention/Naming conventions Test Cases /Output	03(All used)	02 (Partial)	01 (rarely followed)	
4	Post Lab Assignment (03)	03(done well)	2 (Partially Correct)	1(submitted)	
Total					

Signature of the Teacher:

Experiment No: 8

Title: Programming in PROLOG

- Tower of Hanoi
- N-queen & other sample programs

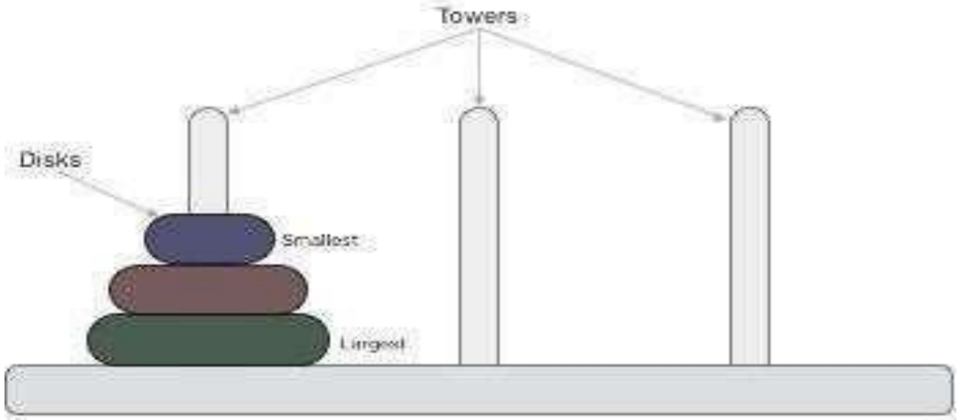
Objective: Solving the tower of Hanoi problem and N-queen using PROLOG

Theory:

A) Tower of Hanoi

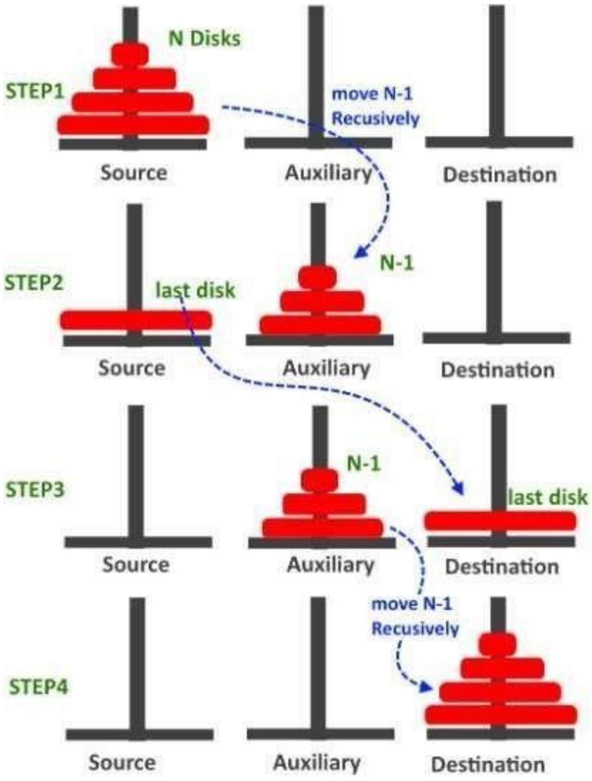
The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. He was inspired by a legend that tells of a Hindu temple where the puzzle was presented to young priests. At the beginning of time, the priests were given three poles and a stack of 64 gold disks, each disk a little smaller than the one beneath it. Their assignment was to transfer all 64 disks from one of the three poles to another, with two important constraints. They could only move one disk at a time, and they could never place a larger disk on top of a smaller one. The priests worked very efficiently, day and night, moving one disk every second.

Tower of Hanoi, is a mathematical puzzle which consists of three towers (pegs) and more than one rings is as depicted –



These rings are of different sizes and stacked upon in an ascending order, i.e., the smaller one sits over the larger one. There are other variations of the puzzle where the number of disks increase, but the tower count remains the same.

Algorithm:



To write an algorithm for Tower of Hanoi, first it needs to learn how to solve this problem with a smaller number of disks, say → 1 or 2. It marks three towers with name, **source**, **destination** and **aux** (only to help moving the disks). If it has only one disk, then it can easily be moved from source to destination peg.

If there are 2 disks −

- First, it moves the smaller (top) disk to aux peg.
- Then, it moves the larger (bottom) disk to the destination peg.

- And finally, it moves the smaller disk from aux to destination peg.

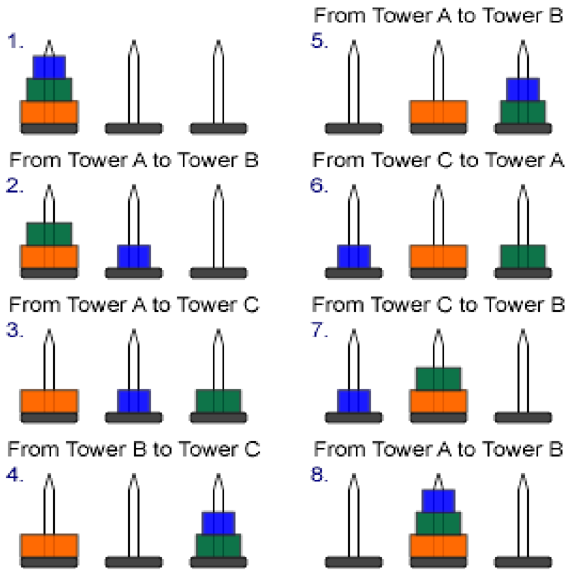
Rules:

The mission is to move all the disks to some other tower without violating the sequence of arrangement. A few rules to be followed for Tower of Hanoi are −

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.

No large disk can sit over a small disk
So now, it is in a position to design an algorithm for Tower of Hanoi with more than two disks. It divides the stack of disks in two parts. The largest disk (nth disk) is in one part and all other (n-1) disks are in the second part.

Following is a representation of solving a Tower of Hanoi puzzle with three diss.



Tower of Hanoi steps performed

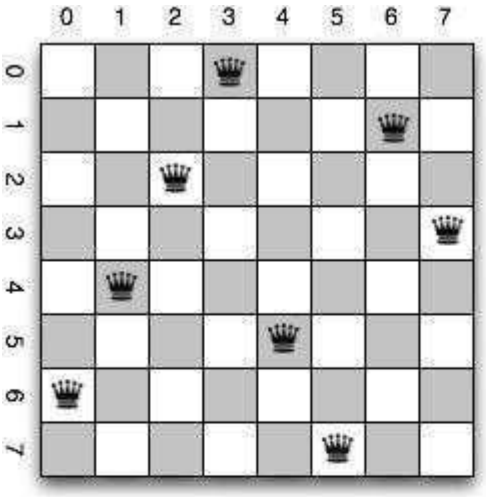
Tower of Hanoi puzzle with n disks can be solved in minimum **2^n–1** steps. This presentation shows that a puzzle with 3 disks has taken **2^3 - 1 = 7** steps.

Our ultimate aim is to move disk **n** from source to destination and then put all other (n1) disks onto it. It can be imagined to apply the same in a recursive way for all given sets of disks.

B) N-queen

Algorithm

```
Step 1. [Initialize N number of queens and insert in Qs]]

                n_queens  (N,  Qs):  -
                    length (Qs, N), Qs
                    ins              1.N,
                    safe_queens(Qs)
                    .

Step 2. [Set safe_queens to null] safe_queens([]).

Step 3. [Take queens Qs and Q and verify if attack is possible]
                safe_queens([Q|Qs]): - safe_queens(Qs, Q, 1),
                safe_queens(Qs).

Step 4. [Continue till Qs id matches N] safe_queens([],
                _, _).
Step 5. [If Q meets no attack declare Q as safe and add to safe_queens]
  [Continue till all queens have been placed/declared safe]
                safe_queens([Q|Qs], Q0, D0:- Q0 #\= Q,
                    abs (Q0 - Q) #\= D0,
                    D1 #= D0 + 1,
  safe_queens (Qs, Q0, D1).
```



CODE:

```
swish1.pl
% Some simple test Prolog programs
% ---------------------------------

% Knowledge bases

loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X, Y) :-
    loves(X, Z),
    loves(Y, Z).
```

```
/** <examples>

?- loves(X, mia).
?- jealous(X, Y).

*/
```

swish2.pl

```
% Define family relationships
parent(terry, john).
parent(terry, mary).
parent(john, pat).
parent(john, anne).
parent(mary, jim).

% Define male and female genders
male(terry).
male(john).
male(pat).
male(jim).
female(mary).
female(anne).

% Define rules for different types of relationships
father(Father, Child) :- parent(Father, Child), male(Father).
mother(Mother, Child) :- parent(Mother, Child), female(Mother).
child(Child, Parent) :- parent(Parent, Child).
grandparent(Grandparent, Grandchild) :- parent(Grandparent, Parent), parent(Parent,
Grandchild).
sibling(Sibling1, Sibling2) :- parent(Parent, Sibling1), parent(Parent, Sibling2),
Sibling1 \= Sibling2.
brother(Brother, Sibling) :- sibling(Brother, Sibling), male(Brother).
sister(Sister, Sibling) :- sibling(Sister, Sibling), female(Sister).

% Define ancestor relationship recursively
ancestor(Ancestor, Descendant) :- parent(Ancestor, Descendant).
ancestor(Ancestor, Descendant) :- parent(Parent, Descendant), ancestor(Ancestor,
Parent).
```

```
% Define descendant relationship recursively
descendant(Descendant, Ancestor) :- ancestor(Ancestor, Descendant).

% Define rules for cousin relationship
cousin(Cousin1, Cousin2) :- parent(Parent1, Cousin1), parent(Parent2, Cousin2),
sibling(Parent1, Parent2), Cousin1 \= Cousin2.

% Define rules for uncle and aunt relationships
uncle(Uncle, NieceNephew) :- parent(Parent, NieceNephew), brother(Uncle, Parent).
aunt(Aunt, NieceNephew) :- parent(Parent, NieceNephew), sister(Aunt, Parent).
```

swish3.pl

```
%%  eliza(+Stimuli, -Response) is det.
%   @param  Stimuli is a list of atoms (words).
%   @author Richard A. O'Keefe (The Craft of Prolog)

eliza(Stimuli, Response) :-
   template(InternalStimuli, InternalResponse),
   match(InternalStimuli, Stimuli),
   match(InternalResponse, Response),
   !.

template([s([i,am]),s(X)], [s([why,are,you]),s(X),w('?')]).
template([w(i),s(X),w(you)], [s([why,do,you]),s(X),w(me),w('?')]).


match([],[]).
match([Item|Items],[Word|Words]) :-
   match(Item, Items, Word, Words).

match(w(Word), Items, Word, Words) :-
   match(Items, Words).
match(s([Word|Seg]), Items, Word, Words0) :-
   append(Seg, Words1, Words0),
   match(Items, Words1).


/** <examples>

?- eliza([i, am, very, hungry], Response).
?- eliza([i, love, you], Response).

*/
```

swish4.pl

```
%%  houses(-Solution)
%   @param  Solution is a list of houses that satisfy all constraints.
%   @author Folklore attributes this puzzle to Einstein
%   @see http://en.wikipedia.org/wiki/Zebra_Puzzle


/* Houses logical puzzle: who owns the zebra and who drinks water?

     1) Five colored houses in a row, each with an owner, a pet, cigarettes, and a drink.
     2) The English lives in the red house.
     3) The Spanish has a dog.
     4) They drink coffee in the green house.
     5) The Ukrainian drinks tea.
     6) The green house is next to the white house.
     7) The Winston smoker has a serpent.
     8) In the yellow house they smoke Kool.
     9) In the middle house they drink milk.
    10) The Norwegian lives in the first house from the left.
    11) The Chesterfield smoker lives near the man with the fox.
    12) In the house near the house with the horse they smoke Kool.
    13) The Lucky Strike smoker drinks juice.
    14) The Japanese smokes Kent.
    15) The Norwegian lives near the blue house.

Who owns the zebra and who drinks water?
*/

% Render the houses term as a nice table.
:- use_rendering(table,
                [header(h('Owner', 'Pet', 'Cigarette', 'Drink', 'Color'))]).

zebra_owner(Owner) :-
    houses(Hs),
    member(h(Owner,zebra,_,_,_), Hs).
```

```
water_drinker(Drinker) :-
    houses(Hs),
    member(h(Drinker,_,_,water,_), Hs).


houses(Hs) :-
    % each house in the list Hs of houses is represented as:
    %      h(Nationality, Pet, Cigarette, Drink, Color)
    length(Hs, 5),                                        %  1
    member(h(english,_,_,_,red), Hs),                     %  2
    member(h(spanish,dog,_,_,_), Hs),                     %  3
    member(h(_,_,_,coffee,green), Hs),                    %  4
    member(h(ukrainian,_,_,tea,_), Hs),                   %  5
    next(h(_,_,_,_,green), h(_,_,_,_,white), Hs),         %  6
    member(h(_,snake,winston,_,_), Hs),                   %  7
    member(h(_,_,kool,_,yellow), Hs),                     %  8
    Hs = [_,_,h(_,_,_,milk,_),_,_],                       %  9
    Hs = [h(norwegian,_,_,_,_)|_],                        % 10
    next(h(_,fox,_,_,_), h(_,_,chesterfield,_,_), Hs),    % 11
    next(h(_,_,kool,_,_), h(_,horse,_,_,_), Hs),          % 12
    member(h(_,_,lucky,juice,_), Hs),                     % 13
    member(h(japanese,_,kent,_,_), Hs),                   % 14
    next(h(norwegian,_,_,_,_), h(_,_,_,_,blue), Hs),      % 15
    member(h(_,_,_,water,_), Hs),                % one of them drinks water
    member(h(_,zebra,_,_,_), Hs).                % one of them owns a zebra

next(A, B, Ls) :- append(_, [A,B|_], Ls).
next(A, B, Ls) :- append(_, [B,A|_], Ls).

/** <examples>

?- zebra_owner(Owner).

?- water_drinker(Drinker).

?- houses(Houses).

*/
```

```
swish5.pl
% render solutions nicely.
:- 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).


/** <examples>

?- queens(8, Queens).

*/
```

swish6.pl

```
% Reading and writing
% -------------------

hello_world :-
    writeln('Hello World!'),
    sleep(1),
    hello_world.

read_and_write :-
    prompt(_, 'Type a term or \'stop\''),
    read(Something),
    (   Something == stop
    ->  true
    ;   writeln(Something),
        read_and_write
    ).


/** <examples>

?- hello_world.
?- read_and_write.

*/
```

swish7.pl

```
% Doing database manipulation
% ---------------------------

:- dynamic p/1.

assert_and_retract :-
    forall(between(1, 10, X), assert(p(X))),
    forall(retract(p(X)), writeln(X)).

assert_many(Count) :-
    forall(between(1, Count, X), assert(p(X))),
    retractall(p(_)).
```

```
/** <examples>

% Basic usage
?- assert_and_retract.

% Show timing
?- assert_many(1 000 000).

% Pengines have a (default) 100Mb limit to their program size
?- assert_many(10 000 000).
*/
```

swish8.pl

```
student(john).
student(mary).
student(peter).
student(emma).

studies(john, math).
studies(john, physics).
studies(mary, biology).
studies(peter, chemistry).
studies(emma, math).

enrolled_in(X, Course) :-
    student(X),
    studies(X, Course).
```

swish9.pl

```
employee(john, developer).
employee(mary, designer).
employee(peter, manager).
employee(emma, analyst).
employee(alex, developer).
employee(sarah, designer).

department(developer, software).
department(designer, creative).
department(manager, administration).
department(analyst, finance).
```

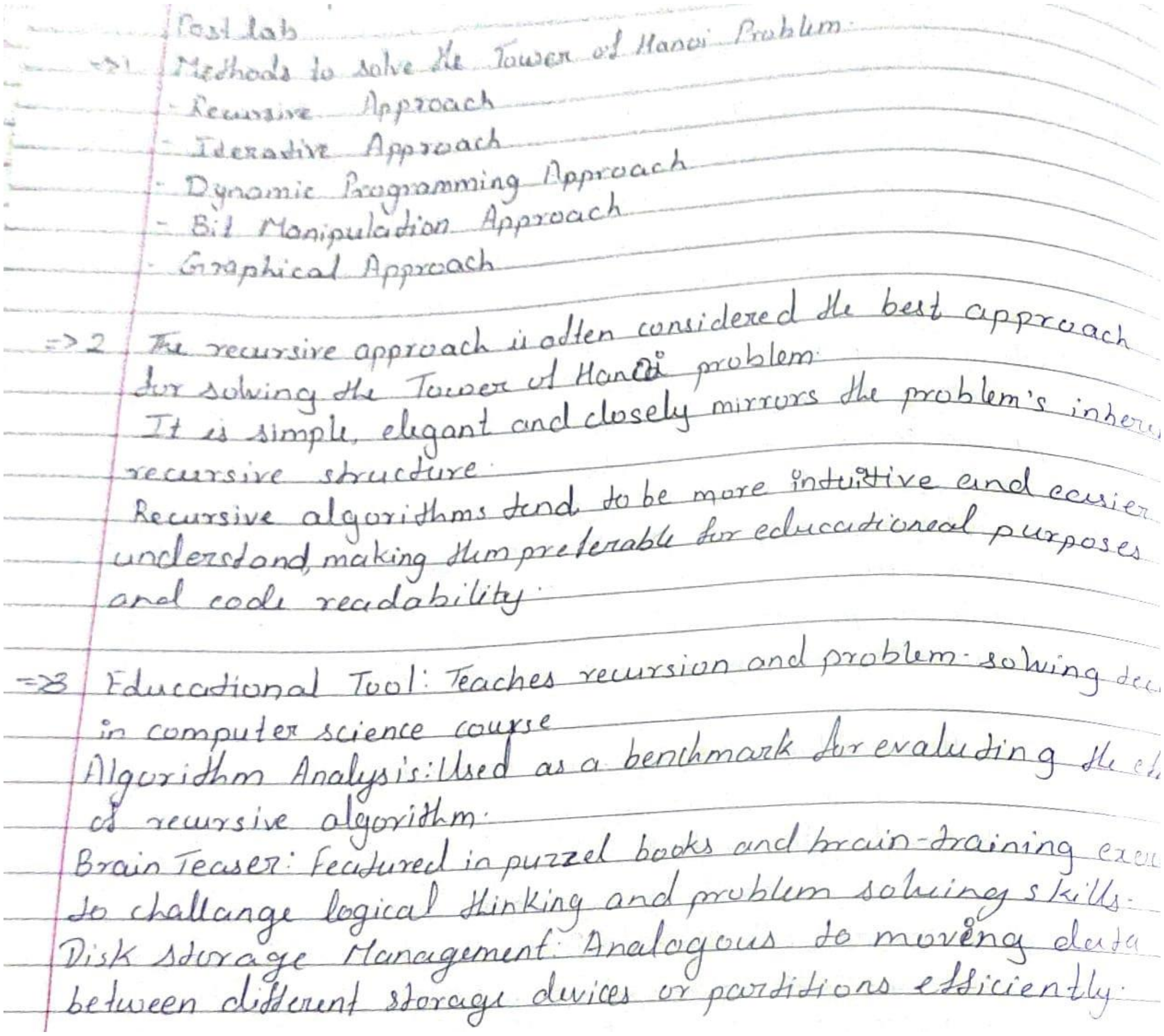
```
works_in(X, Department) :-
    employee(X, Role),
    department(Role, Department).
```

swish10.pl

```
animal(cat).
animal(dog).
animal(elephant).
animal(giraffe).

mammal(cat).
mammal(dog).
mammal(elephant).
mammal(giraffe).

has_four_legs(cat).
has_four_legs(dog).
has_four_legs(elephant).

has_long_neck(giraffe).
```

Post Lab Questions:

1. List all the methods which could be used to solve the tower of Hanoi problem.
2. Which is the best approach and why?
3. What are the applications of the Tower of Hanoi?



Post lab

=>1 Methods to solve the Tower of Hanoi Problem:
- Recursive Approach
- Iderative Approach
- Dynamic Programming Approach
- Bit Manipulation Approach
- Graphical Approach

=>2 The recursive approach is often considered the best approach for solving the Tower of Hanoi problem.
It is simple, elegant and closely mirrors the problem's inher
recursive structure.
Recursive algorithms tend to be more intuitive and easier understand, making them preferable for educational purposes and code readability.

=>3 Educational Tool: Teaches recursion and problem-solving dec
in computer science course
Algorithm Analysis: Used as a benchmark for evaluating the ef
of recursive algorithm.
Brain Teaser: Featured in puzzel books and brain-training exe
to challange logical thinking and problem solving skills.
Disk storage Management: Analogous to moving data between different storage devices or partitions efficiently.