

**INSTITUTE OF TECHNOLOGY AND MANAGEMENT  
GIDA GORAKHPUR**



**BRANCH: COMPUTER SCIENCE AND ENGINEERING  
SUBJECT: ARTIFICIAL INTELLIGENCE LAB  
LAB CODE: (RCS752)  
CSE-B41**

**SUBMITTED BY:  
ABHISHEK MISHRA  
ROLL NO: 1712010006**

**SUBMITTED TO:  
MR. ANUJ KUMAR**

**GRADE:**

**SIGNATURE:**

# INDEX

S. No.	Practical's Name	Date	Remark
1	Study of Prolog.		
2	Write simple fact for the statements using PROLOG.		
3	Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.		
4	Write a program to solve the Monkey Banana problem.		
5	WAP in turbo prolog for medical diagnosis and show the advantage and disadvantage of green and red cuts.		
6	WAP to implement factorial, Fibonacci of a given number.		
7	Write a program to solve 4-Queen problem.		
8	Write a program to solve traveling salesman problem.		
9	Write a program to solve water jug problem using Prolog.		

# Experiment Number: 1

## OBJECTIVE: Study of Prolog.

### PROLOG-PROGRAMMING IN LOGIC

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), Marriten van Emden at Edinburgh (experimental demonstration) and Alian 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 centred around 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.

### SYMBOLIC LANGUAGE

PROLOG is a programming language for symbolic, non-numeric computation. It is especially well suited for solving problems that involve objects and relations between objects. For example, it is an easy exercise in prolog to express spatial relationship between objects, such as the blue sphere is behind the green one. It is also easy to state a more general rule: if object X is closer to the observer than object Y. and object Y is closer than Z, then X must be closer than Z. PROLOG can reason about the spatial relationships and their consistency with respect to the general rule. Features like this make PROLOG a powerful language for Artificial LanguageAI,) and non- numerical programming.

There are well-known examples of symbolic computation whose implementation in other standard languages took tens of pages of indigestible code, when the same algorithms were implemented in PROLOG, the result was a crystal-clear program easily fitting on one page.

### FACTS, RULES AND QUERIES

Progmpping in PROIOG is accomplished by creating a database of facts and rules about objects, their properties, and their relationships to other objects. Queries then can be posed about the objects and valid conclusions will be determined and returned by the program Responses to user queries are determined through a form of inference control known as resolution.

#### a) FACTS:

Some facts about family relationships could be written as:

```
sister( sue,bill)
parent( ann.sam)
male(jo)
female( riya)
```

#### b) RULES:

To represent the general rule for grandfather, we write:

```
grand f.gher( X2)
parent(X,Y)
parent( Y,Z)
male(X)
```

#### c) QUERIES:

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
```

## **PROLOG IN DESIGNING EXPERT SYSTEMS**

An expert system is a set of programs that manipulates encoded knowledge to solve problems in a specialized domain that normally requires human expertise. An expert system's knowledge is obtained from expert sources such as texts, journal articles, databases etc and encoded in a form suitable for the system to use in its inference or reasoning processes. Once a sufficient body of expert knowledge has been acquired, it must be encoded in some form, loaded into knowledge base, then tested, and refined continually throughout the life of the system. PROLOG serves as a powerful language in designing expert systems because of its following features.

- Use of knowledge rather than data
- Modification of the knowledge base without recompilation of the control programs.
- Capable of explaining conclusion.
- Symbolic computations resembling manipulations of natural language.
- Reason with meta-knowledge.

## **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.

## Experiment Number: 2

**OBJECTIVE:** Write simple fact for following:

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

**Program:**

**Clauses:**

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

**Output:**

?-likes(ram,What).  
What= mango  
?-likes(Who,cindy).  
Who= cindy  
?-red(What).  
What= rose  
?-owns(Who,What).  
Who= john  
What= gold.

## Experiment Number: 3

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

**Program:**

**Production rules:**

$c\_to\_f(F,C) \rightarrow F \text{ is } C * 9 / 5 + 32$

$freezing \rightarrow f < = 32$

**Clauses:**

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

$F \text{ is } C * 9 / 5 + 32.$

$freezing(F) :-$

$F \leq 32.$

**Output:-**

```
% c:/users/hp/documents/prolog/akm (2) compiled 0.00 sec, -2 clauses
?- c_to_f(100,X).
X = 212.

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

?- freezing(15).
true.

?- freezing(45).
false.
```

## Experiment Number: 4

**OBJECTIVE:** Write a program to solve the Monkey Banana problem.

**Description:-**Imagine a room containing a monkey, chair and some bananas. That has been hanged from the centre 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 wants it, but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use. 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.

**Clauses:**

```
on(floor,monkey).
on(floor,chair).
in(room,monkey).
in(room,chair).
in(room,banana).
at(ceiling,banana).
strong(monkey).
grasp(monkey).
climb(monkey,chair).
push(monkey,chair):-strong(monkey).
under(banana,chair):-push(monkey,chair).
canreach(banana,monkey):-at(floor,banana);at(ceiling,banana),under(banana,chair),
climb(monkey,chair).
canget(banana,monkey):-canreach(banana,monkey),grasp(monkey).
```

**Output:**

```
% c:/users/hp/documents/prolog/monkey compiled 0.00 sec, 0 clauses
?- canget(banana,monkey).
true.

?- canget(banana,chair).
false.

?- canget(X,Y).
X = banana,
Y = monkey.

?-
```

## Experiment Number: 5

**OBJECTIVE: WAP in turbo prolog for medical diagnosis and show the advantage and disadvantage of green and red cuts.**

**Program:**

**Domains:**

disease,indication=symbol

name-string

**Predicates:**

hypothesis(name,disease)

symptom(name,indication)

response(char)

go

goonce

**clauses:**

go:-

goonce

write("will you like to try again (y/n)?"),

response(Reply),

Reply='n'.

go.

goonce:-

write("what is the patient's name"),nl,

readln(Patient),

hypothesis(Patient,Disease),!,

write(Patient,"probably has",Disease),!,

goonce:-

write("sorry, i am not in a position to diagnose"),

write("the disease").

symptom(Patient,fever):-

write("does",Patient,"has a fever (y/n)?"),nl,

response(Reply),

Reply='y',nl.

symptom(Patient,rash):-

write("does",Patient,"has a rash (y/n)?"),nl,

response(Reply),

Reply='y',

symptom(Patient\_body,ache):-

write("does",Patient,"has a body ache

(y/n)?"),nl, response(Reply).

Reply='y',nl.

symptom(Patient,runny\_nose):-

write("does",Patient,"has a runny\_nose (y/n)?"),

response(Reply),

Reply='y'

hypothesis(Patient,flu):-

symptom(Patient,fever),

symptom(Patient,body\_ache),

hypothesis(Patient,common\_cold):-



```
symptom(Patient,body_ache),  
Symptom(Patient,runny_nose).  
response(Reply):-  
readchar(Reply),  
write(Reply).
```

**Output:**

```
makewindow(1,7,"Expert Medical Diagnosis",2,2,23,70), go.
```

## Experiment Number: 6

**OBJECTIVE:** WAP to implement factorial, Fibonacci of a given number.

**Program:**

**Factorial:**

```
factorial(0,1).  
factorial(N,F) :-  
    N>0,  
    N1 is N-1,  
    factorial(N1,F1),  
    F is N * F1.
```

**Output:**

```
% c:/users/hp/documents/prolog/fact compiled 0.00 sec, 0 clauses  
?- factorial(4,X).  
X = 24 ,  
  
?-
```

**Fibonacci:**

```
fib(0, 0).  
fib(X, Y) :- X > 0, fib(X, Y, _).  
fib(1, 1, 0).  
fib(X, Y1, Y2) :-  
    X>1,  
    X1 is X - 1,  
    fib(X1, Y2, Y3),  
    Y1 is Y2 + Y3.
```

**Output:**

```
% c:/users/hp/documents/prolog/akmm compiled 0.00 sec, 0 clauses  
?- fib(10,X).  
X = 55 ,  
  
?-
```

## Experiment Number: 7

**OBJECTIVE:** Write a program to solve 4-Queen problem.

**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.

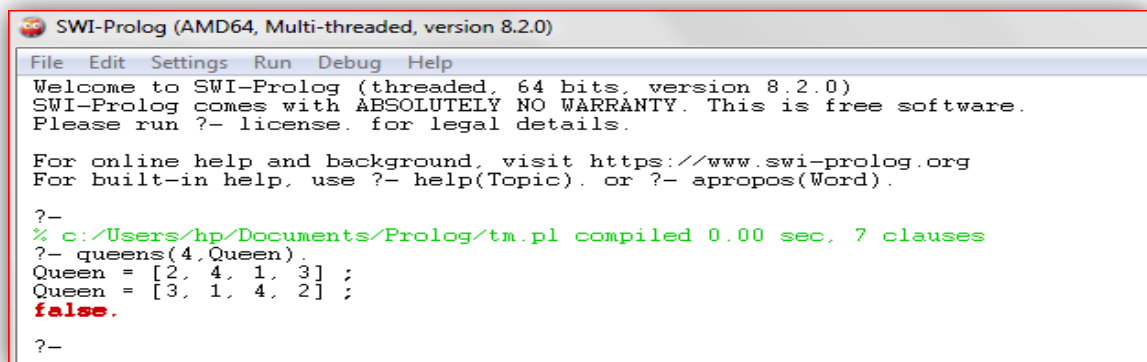
	1	2	3	4
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6
4	4	5	6	7

**Fig1: The n Queens Chessboard.**

**Program:-**

```
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).
```

**Output:**



```
SWI-Prolog (AMD64, Multi-threaded, version 8.2.0)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.2.0)
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).

?-
% c:/Users/hp/Documents/Prolog/tm.pl compiled 0.00 sec, 7 clauses
?- queens(4,Queen).
Queen = [2, 4, 1, 3] ;
Queen = [3, 1, 4, 2] ;
false.
?-
```

## Experiment Number: 8

**OBJECTIVE:** Write a program to solve traveling salesman problem.

**Dataset:**

```
edge(a, b, 3).
edge(a, c, 4).
edge(a, d, 2).
edge(a, e, 7).
edge(b, c, 4).
edge(b, d, 6).
edge(b, e, 3).
edge(c, d, 5).
edge(c, e, 8).
edge(d, e, 6).
edge(b, a, 3).
edge(c, a, 4).
edge(d, a, 2).
edge(e, a, 7).
edge(c, b, 4).
edge(d, b, 6).
edge(e, b, 3).
edge(d, c, 5).
edge(e, c, 8).
edge(e, d, 6).
edge(a, h, 2).
edge(h, d, 1).
```

**Program:**

```
len([], 0).
len([H|T], N):- len(T, X), N is X+1 .
best_path(Visited, Total):- path(a, a, Visited, Total).
path(Start, Fin, Visited, Total) :- path(Start, Fin, [Start], Visited, 0, Total).
path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-edge(Start, StopLoc, Distance), NewCostn is
Costn + Distance, \+ member(StopLoc, CurrentLoc),path(StopLoc, Fin, [StopLoc|CurrentLoc],
Visited, NewCostn, Total).
path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-edge(Start, Fin, Distance),
reverse([Fin|CurrentLoc], Visited), len(Visited, Q),(Q\=7 -> Total is 100000; Total is Costn +
Distance).
shortest_path(Path):-setof(Cost-Path, best_path(Path,Cost), Holder),pick(Holder,Path).
best(Cost-Holder,Bcost-_,Cost-Holder):- Cost<Bcost,!.
best(_,X,X).
pick([Cost-Holder|R],X):- pick(R,Bcost-Bholder),best(Cost-Holder,Bcost-Bholder,X),!.
pick([X],X).
```

**Output:**

```
Warning: c:/users/hp/documents/prolog/jug.pl:24:
Warning: Singleton variables: [H]
% c:/users/hp/documents/prolog/jug compiled 0.03 sec, 0 clauses
?- shortest_path(Path).
Path = 20-[a, h, d, e, b, c, a].
?-
```

## Experiment Number: 9

**OBJECTIVE:** Write a program to solve water jug problem using Prolog.

**Water Jug Problem:** You are given two jugs, a 4lit one and a 3lit one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2lit of water in the 4lit jug?

**Program:**

```
start(2,0):-write(' 4lit Jug:  2 | 3lit Jug:  0\n'),
            write('~~~~~\n'),
            write('Goal Reached! Congrats!!\n'),
            write('~~~~~\n').
start(X,Y):-write(' 4lit Jug:  '),write(X),write('| 3lit Jug:  '),
            write(Y),write('\n'),
            write(' Enter the move::'),
            read(N),
            contains(X,Y,N).

contains(_,Y,1):-start(4,Y).
contains(X,_,2):-start(X,3).
contains(_,Y,3):-start(0,Y).
contains(X,_,4):-start(X,0).
contains(X,Y,5):-N is Y-4+X, start(4,N).
contains(X,Y,6):-N is X-3+Y, start(N,3).
contains(X,Y,7):-N is X+Y, start(N,0).
contains(X,Y,8):-N is X+Y, start(0,N).

main():-write(' Water Jug Game \n'),
        write('Intial State: 4lit Jug- 0lit\n'),
        write('          3lit Jug- 0lit\n'),
        write('Final State: 4lit Jug- 2lit\n'),
        write('          3lit Jug- 0lit\n'),
        write('Follow the Rules: \n'),
        write('Rule 1: Fill 4lit Jug\n'),
        write('Rule 2: Fill 3lit Jug\n'),
        write('Rule 3: Empty 4lit Jug\n'),
        write('Rule 4: Empty 3lit Jug\n'),
        write('Rule 5: Pour water from 3lit Jug to fill 4lit Jug\n'),
        write('Rule 6: Pour water from 4lit Jug to fill 3lit Jug\n'),
        write('Rule 7: Pour all of water from 3lit Jug to 4lit Jug\n'),
        write('Rule 8: Pour all of water from 4lit Jug to 3lit Jug\n'),
        write(' 4lit Jug:  0 | 3lit Jug:  0'),nl,
        write(' Enter the move::'),
        read(N),nl,
        contains(0,0,N).
```

## Output:

```
?- main().
  Water Jug Game
Initial State: 4lit Jug- 0lit
               3lit Jug- 0lit
Final State:  4lit Jug- 2lit
               3lit Jug- 0lit
Follow the Rules:
Rule 1: Fill 4lit Jug
Rule 2: Fill 3lit Jug
Rule 3: Empty 4lit Jug
Rule 4: Empty 3lit Jug
Rule 5: Pour water from 3lit Jug to fill 4lit Jug
Rule 6: Pour water from 4lit Jug to fill 3lit Jug
Rule 7: Pour all of water from 3lit Jug to 4lit Jug
Rule 8: Pour all of water from 4lit Jug to 3lit Jug
  4lit Jug:  0 | 3lit Jug:  0
Enter the move::1.

  4lit Jug:  4 | 3lit Jug:  0|
Enter the move::|: 6.
  4lit Jug:  1 | 3lit Jug:  3|
Enter the move::|: 4.
  4lit Jug:  1 | 3lit Jug:  0|
Enter the move::|: 8.
  4lit Jug:  0 | 3lit Jug:  1|
Enter the move::|: 1.
  4lit Jug:  4 | 3lit Jug:  1|
Enter the move::|: 6.
  4lit Jug:  2 | 3lit Jug:  3|
Enter the move::|: 4.
  4lit Jug:  2 | 3lit Jug:  0|
~~~~~
Goal Reached! Congrats!!
~~~~~
true .
```