

Jawaharlal Nehru Engineering College
Aurangabad

Laboratory Manual

Artificial Intelligence

For

Final Year Students CSE

Dept: Computer Science & Engineering

© Author JNEC, Aurangabad

FOREWORD

It is my great pleasure to present this laboratory manual for **FINAL YEAR COMPUTER SCIENCE & ENGINEERING** students for the subject of Artificial Intelligence. As a student, many of you may be wondering about the subject and exactly that has been tried through this manual.

As you may be aware that MGM has already been awarded with ISO 9000 certification and it is our aim to technically equip students taking the advantage of the procedural aspects of ISO 9000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself will relieve them in future as much of the load will be taken care by the enthusiastic energies of the students once they are conceptually clear.

Dr. S.D. Deshmukh

Principal

LABORATORY MANUAL CONTENTS

This manual is intended for FIANL YEAR COMPUTER SECINCE & ENGINEERING students for the subject of **Artificial Intelligence**. This manual typically contains practical/Lab Sessions related Artificial Intelligence covering various aspects related the subject to enhanced understanding.

This course will also helpful for student to understanding and design expert system. We have made the efforts to cover various aspects of the subject covering these labs encompass the regular materials as well as some advanced experiments useful in real life applications.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

Dr. V.B. Musande
HOD, CSE Dept

MS. A.H. Telgaonkar
Lecturer, CSE Dept



Jawaharlal Nehru Engineering College, Aurangabad

Department of Computer Science and Engineering

Vision of CSE Department

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

Mission of the CSE Department:

- Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.
- Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

Programme Educational Objectives

Graduates will be able to

- I.** To analyze, design and provide optimal solution for Computer Science & Engineering and multidisciplinary problems.
- II.** To pursue higher studies and research by applying knowledge of mathematics and fundamentals of computer science.
- III.** To exhibit professionalism, communication skills and adapt to current trends by engaging in lifelong learning.

Programme Outcomes (POs):

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage independent and life-long learning in the broadest context of technological change.

DOs and DON'Ts in Laboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.
2. All the students should sit according to their roll numbers starting from their left to right.
3. All the students are supposed to enter the terminal number in the log book.
4. Do not change the terminal on which you are working.
5. All the students are expected to get at least the algorithm of the program/concept to be implemented.
6. Strictly observe the instructions given by the teacher/Lab Instructor.

Instruction for Laboratory Teachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.
2. Students should be taught for taking the printouts under the observation of lab teacher.
3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

SUBJECT INDEX

Sr. No.	Title	Page No.
1	Study of prolog	6-9
2	Program to generate family tree	10-16
3	Program for water jug problem solving	17-20
4	Program checking a person eligible for voting	21-22
5	Program to calculate factorial of number	23-25
6	Program for generating Fibonacci series	26-28
7	Program for generating pyramid	30-34
8	Program for towers of Hanoi puzzle	35-40
9	Design an expert system	41-46

1. Study of Prolog

Objective:

- Learn basics of Prolog language.
- How to write the Prolog program in SWI-Prolog.
- How to create the knowledge base in prolog.

Description:

- This is program for getting knowledge about how to create a Prolog program in SWI-Prolog IDE.
- In this program we learn how to create a facts in prolog.

What is Fact?

A fact is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be universally quantified.

Note that every fact end with (.) dot symbol.

Ex-

```
likes(john, susie).           /* John likes Susie */
likes(X, susie).             /* Everyone likes Susie */
likes(john, Y).              /* John likes everybody */
```

What is Rule?

A **rule** is a predicate expression that uses logical implication (:-) to describe a relationship among facts. Thus a Prolog rule takes the form

```
left_hand_side :- right_hand_side .
```

This sentence is interpreted as: *left_hand_side* if *right_hand_side*. The *left_hand_side* is restricted to a single, positive, literal, which means it must consist of a positive atomic expression. It cannot be negated and it cannot contain logical connectives.

This notation is known as a Horn clause. In Horn clause logic, the left hand side of the clause is the conclusion, and must be a single positive literal. The right hand side contains the premises. The Horn clause calculus is equivalent to the first-order predicate calculus.

Examples of valid rules:

```
friends(X,Y) :- likes(X,Y),likes(Y,X).
               /* X and Y are friends if they like each other */
hates(X,Y) :- not(likes(X,Y)).
               /* X hates Y if X does not like Y. */
```


What is Query?

The Prolog interpreter responds to queries about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" and displays any variable bindings that it made in coming up with the answer. If it fails to prove the query true, it answers "No".

Ex-

```
father_of(joe,paul).
father_of(joe,mary).
mother_of(jane,paul).
mother_of(jane,mary).
male(paul).      male(joe).
female(mary). female(jane).
```

Program 1.Simple program on How to create the facts in Prolog Program.

Facts: - facts defined in program are -

```
awesomегirl(Seema).
```

```
awesomегirl(neha).
```

```
awesomегirl(jeni).
```

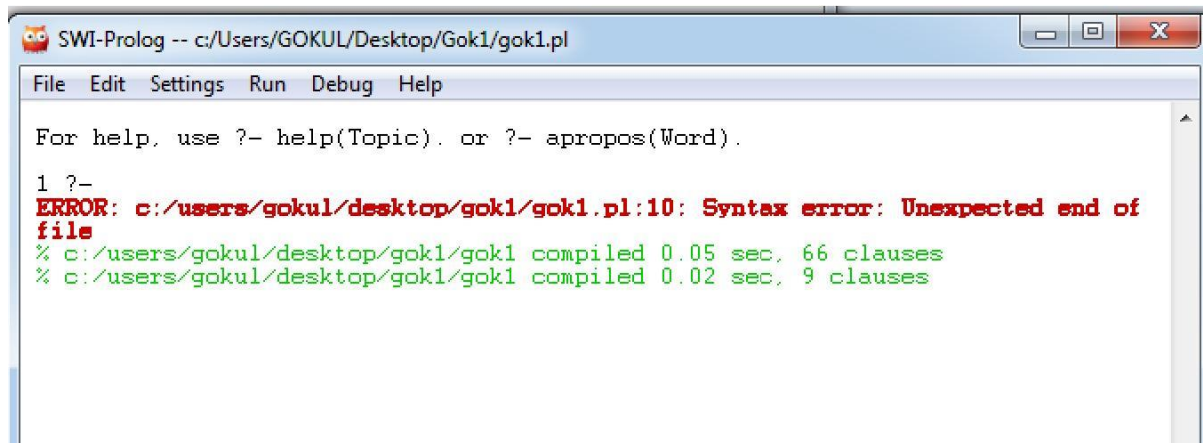
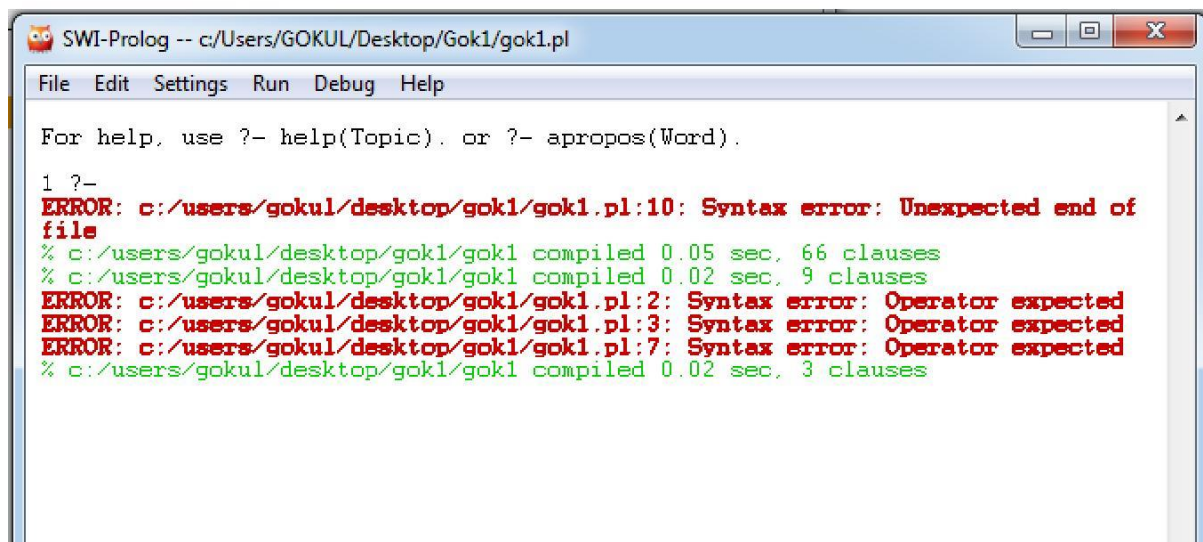
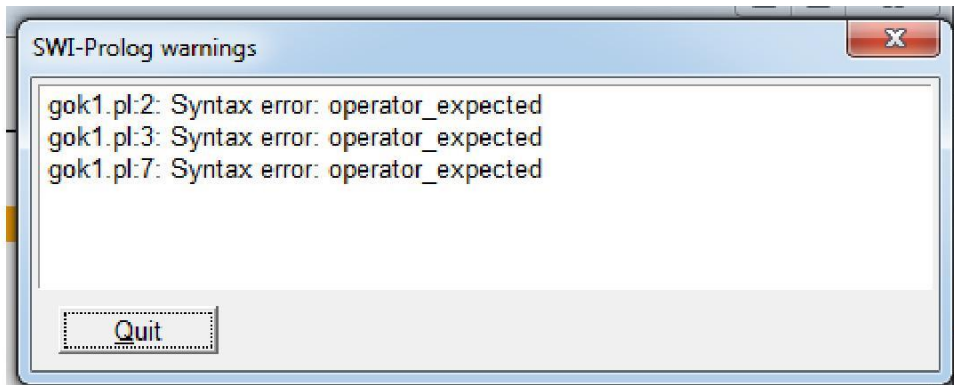
```
loves(kapil,jeni).
```

```
loves(raju,jeni).
```

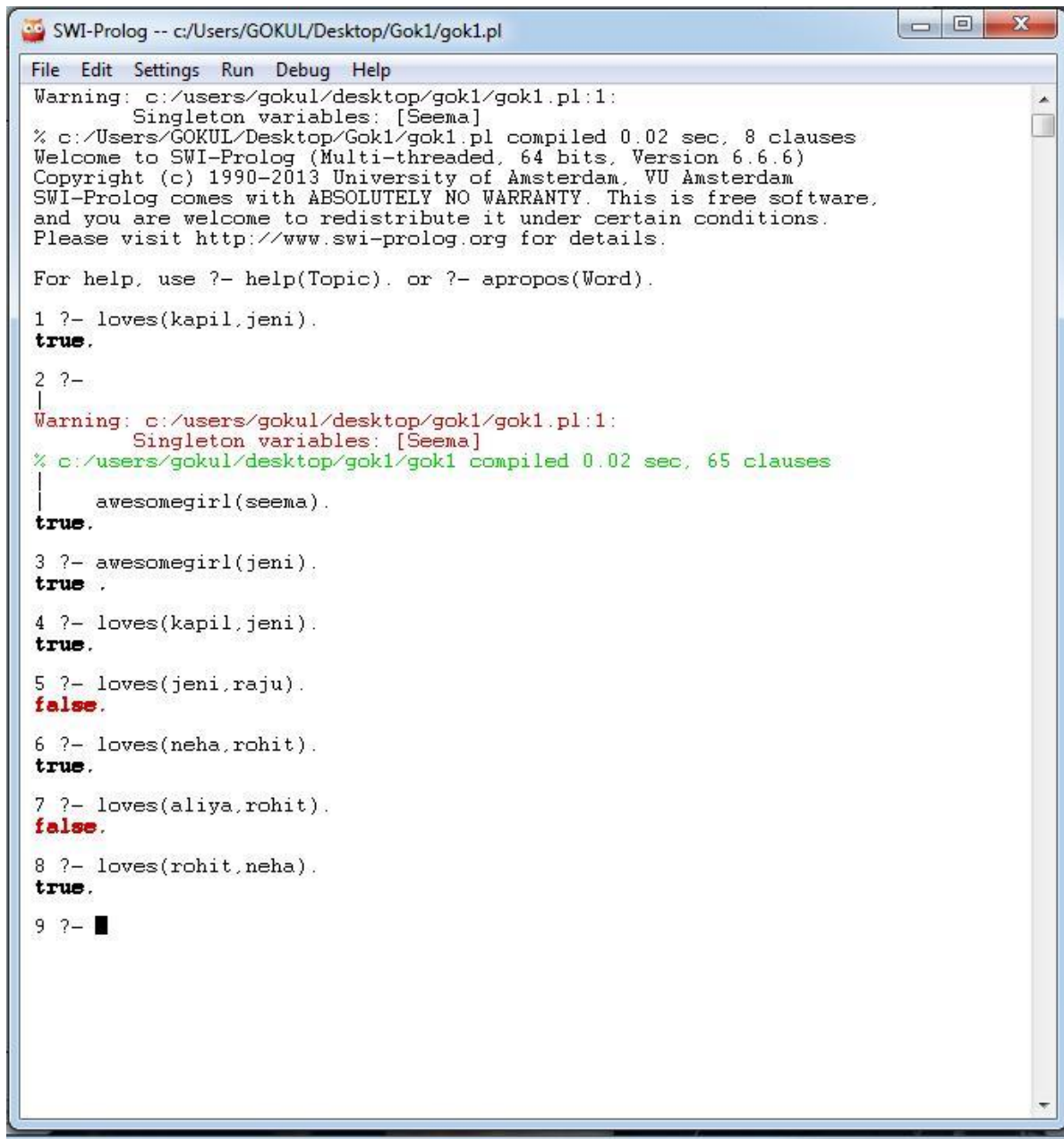
```
loves(rohit,neha).
```

```
loves(neha,rohit).
```

Errors:



Results and Conclusion:



```
SWI-Prolog -- c:/Users/GOKUL/Desktop/Gok1/gok1.pl
File Edit Settings Run Debug Help
Warning: c:/users/gokul/desktop/gok1/gok1.pl:1:
Singleton variables: [Seema]
% c:/Users/GOKUL/Desktop/Gok1/gok1.pl compiled 0.02 sec, 8 clauses
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- loves(kapil,jeni).
true.

2 ?-
|
Warning: c:/users/gokul/desktop/gok1/gok1.pl:1:
Singleton variables: [Seema]
% c:/users/gokul/desktop/gok1/gok1 compiled 0.02 sec, 65 clauses
|
| awesomegirl(seema).
true.

3 ?- awesomegirl(jeni).
true.

4 ?- loves(kapil,jeni).
true.

5 ?- loves(jeni,raju).
false.

6 ?- loves(neha,rohit).
true.

7 ?- loves.aliya.rohit).
false.

8 ?- loves(rohit,neha).
true.

9 ?- █
```

2. Program to generate family tree

Objective:

- Learn how to create a knowledge base
- Learn how to define rules in prolog
- How to use Prolog

Description:

Family tree is a simple tree that maintains the record of family members.

Following tree shows the example of family tree-

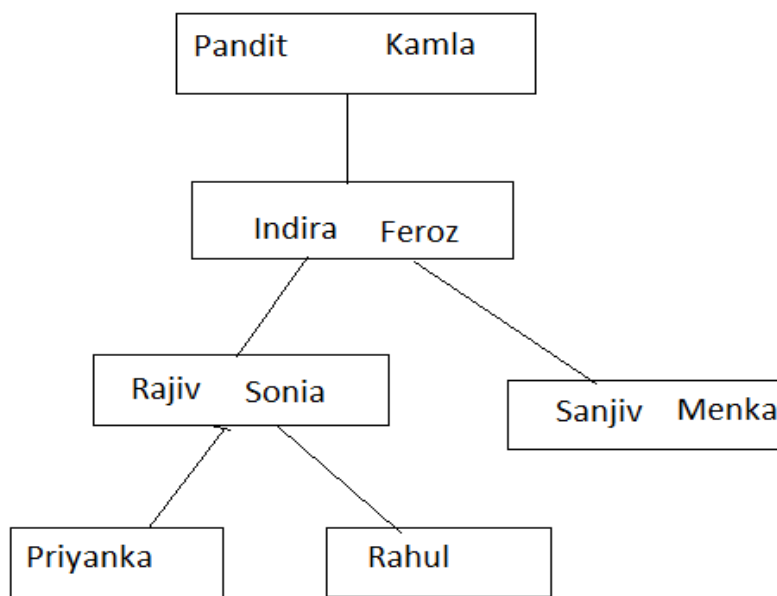


Fig. Family Tree

Following is a program to generate family tree.

Facts:

male(pandit).

male(feroz).

male(rajiv).

male(sanjiv).

male(rahul).

female(kamla).

female(indira).

female(menka).

female(sonia).

female(priyanka).

child(indira,pandit).

child(indira,kamla).

child(rajiv,feroz).

child(rajiv,indira).

child(sanjiv,feroz).

child(sanjiv,indira).

child(priyanka,rajiv).

child(priyanka,sonia).

child(rahul,sonia). child(rahul,rajiv).

Rules:

parent(Y,X):- child(X,Y).

father(X,Y):- male(X),parent(X,Y).

mother(X,Y):- female(X),parent(X,Y).

fathersb(X,Y):-father(Z,X),father(Z,Y),mother(W,X),mother(W,Y),(X\==Y).

brother(X,Y):-fathersb(X,Y),male(X).

sister(X,Y):-fathersb(X,Y),female(X).

son(X):-fathersb(X,Y),male(X).

daughter(X):-fathersb(X,Y),female(X).

grandparent(X,Z):-parent(X,Y),parent(Y,Z).

Errors:

1 ?-

ERROR: c:/users/gokul/desktop/ai/gok1/familytree.pl:1: Syntax error: Operator expected

Warning: c:/users/gokul/desktop/ai/gok1/familytree.pl:27:

Singleton variables: [Y]

Warning: c:/users/gokul/desktop/ai/gok1/familytree.pl:28:

Singleton variables: [Y]

% c:/users/gokul/desktop/ai/gok1/familytree compiled 0.02 sec, 85 clauses

1 ?-

| fMale(X).

ERROR: toplevel: Undefined procedure: fMale/1 (DWIM could not correct goal)

2 ?-

| child(X,Y,Z).

ERROR: Undefined procedure: child/3

ERROR: However, there are definitions for:

ERROR: child/2

false.

3 ?-

| son(X).

X = rahul .

Results and Conclusion:

Output:

% c:/users/gokul/desktop/ai/gok1/familytree compiled 0.02 sec, 87 clauses

1 ?- child(X,Y).

X = indira,

Y = pandit ;

X = indira,

Y = kamla ;

X = rajiv,

Y = feroz ;

X = rajiv,

Y = indira ;

X = sanjiv,

Y = feroz ;

X = sanjiv,

Y = indira ;

X = priyanka,

Y = rajiv ;

X = priyanka,

Y = sonia ;

X = rahul,

Y = sonia ;

X = rahul,

Y = rajiv.

2 ?- parent(X,Y).

X = pandit,

Y = indira ;

X = kamla,

Y = indira ;

X = feroz,

Y = rajiv ;

X = indira,

Y = rajiv ;

X = feroz,

Y = sanjiv ;

X = indira,

Y = sanjiv ;

X = rajiv,

Y = priyanka ;

X = sonia,

Y = priyanka ;

X = sonia,

Y = rahul ;

X = rajiv,

Y = rahul.

3 ?- father(X,Y).

X = pandit,

Y = indira ;

X = feroz,

Y = rajiv ;

X = feroz,

Y = sanjiv ;

X = rajiv,

Y = priyanka ;

X = rajiv,

Y = rahul ;

false.

4 ?- mother(X,Y).

X = kamla,

Y = indira .

5 ?-

| mother(X,Y).

X = kamla,

Y = indira ;

X = indira,

Y = rajiv ;

X = indira,

Y = sanjiv ;

X = sonia,

Y = priyanka ;

X = sonia,

Y = rahul ;

false.

6 ?- son(X).

X = rajiv ;

X = sanjiv ;

X = rahul ;

false.

7 ?- daughter(X).

X = priyanka ;

false.

8 ?- grandparent(X,Y).

X = pandit,

Y = rajiv ;

X = pandit,

Y = sanjiv ;

X = kamla,

Y = rajiv ;

X = kamla,

Y = sanjiv ;

X = feroz,

Y = priyanka ;

X = feroz,

Y = rahul ;

X = indira,

Y = priyanka ;

X = indira,

Y = rahul ;

false.

9 ?-

3. Program for Water Jug Problem

Objective:

- Learn basic login programming.
- How to use logical operators in prolog.
- Learn to solve real world problem using prolog.
- Learn some special operators in prolog like nl, is, etc.
- Learn and understand the use of write function for printing.

Description:

In water-jug problem we have two jugs of the capacity of 3 liter and 4 liter. None of jug has any marker on it to check how much water it contain and also not any of unit of measurement to check how much water available in jug. Problem is to fill the 4 liter jug with exactly 2 liter water. Suppose jug X is of 4 liter and Y is of 3 liter. i.e. X=4 liter & Y=3 liter. Initial state is (X,Y)=(0,0) Goal state is (X,Y)=(2,n) We have a unlimited supply of water and we have to fill the jug X with 2 liter water. Following is a program for the solution for this problem. Using prolog so many real problems can be solved.

Program:

```
jug(A,B):- A is 0, B is 0,
nl, write('fill A Goto Jug(0,3)'), jug(0,3).
jug(A,B):- A is 0, B is 3,
nl, write('fill A goto Jug(3,0)'),jug(3,0).
jug(A,B):- A is 3, B is 0,
nl, write('fill B Goto Jug(3,3)'),jug(3,3).
jug(A,B):-A is 3, B is 3,
nl, write('fill A Goto Jug(4,2)'),jug(4,2).
jug(A,B):- A is 4, B is 2,
nl, write('Empty A Goto Jug(0,2)'),jug(0,2).
jug(A,B):- A is 0, B is 2,
nl, write('Transfer from B to A Goto Jug(2,0)'),jug(2,0).
```

```

jug(A,B):- A is 4, B is 0,
nl, write('fill B Goto Jug(1,3)'),jug(1,3).
jug(A,B):- A is 1, B is 3,
nl, write('Empty B Goto Jug(1,0)'),jug(1,0).
jug(A,B):- A is 1, B is 0,
nl, write('Transfer from A to B Goto Jug(0,1)'),jug(0,1).
jug(A,B):- A is 0, B is 1,
nl, write('fill A Goto Jug(4,1)'),jug(4,1).
jug(A,B):- A is 4, B is 1,
nl, write('fill B Goto Jug(2,2)'),jug(2,2).
jug(A,B):- A is 2, B is 2,
nl, write('Empty B Goto Jug(2,0)'),jug(2,0).
jug(A,B):- A is 2, B is 0,
nl, write('Task Completed').

```

Errors:

```
% c:/users/gokul/desktop/ai/ggg compiled 0.00 sec, 14 clauses
```

```
1 ?- jug(4,0).
```

```
fill B Goto Jug(1,3)
```

```
ERROR: jug/2: Undefined procedure: jug1/2
```

```
ERROR: However, there are definitions for:
```

```
ERROR: jug/2
```

```
Exception: (7) jug1(1, 3) ? creep
```

```
Exception: (6) jug(4, 0) ? creep
```

```
2 ?-
```

```
% c:/users/gokul/desktop/ai/ggg compiled 0.00 sec, 14 clauses
```

```
ERROR: c:/users/gokul/desktop/ai/ggg.pl:44: Syntax error: String too long (see style_check/1)
```

```
% c:/users/gokul/desktop/ai/ggg compiled 0.03 sec, 62 clauses
```

Result and Conclusion:

% c:/users/gokul/desktop/ai/ggg compiled 0.00 sec, 14 clauses

2 ?- jug(0,0).

fill A Goto Jug(0,3)

fill A goto Jug(3,0)

fill B Goto Jug(3,3)

fill A Goto Jug(4,2)

Empty A Goto Jug(0,2)

Transfer from B to A Goto Jug(2,0)

Task Completed

true

3 ?-

|

| jug(3,0).

fill B Goto Jug(3,3)

fill A Goto Jug(4,2)

Empty A Goto Jug(0,2)

Transfer from B to A Goto Jug(2,0)

Task Completed

true

[1] 4 ?- jug(2,2).

Empty B Goto Jug(2,0)

Task Completed

true

[2] 5 ?- jug(4,3).

false.

[2] 6 ?- jug(3,3).

fill A Goto Jug(4,2)

Empty A Goto Jug(0,2)

Transfer from B to A Goto Jug(2,0)

Task Completed

true

[2] 7 ?- jug(0,0).

fill A Goto Jug(0,3)

fill A goto Jug(3,0)

fill B Goto Jug(3,3)

fill A Goto Jug(4,2)

Empty A Goto Jug(0,2)

Transfer from B to A Goto Jug(2,0)

Task Completed

true .

[2] 8 ?-

| jug(1,2).

false.

[2] 9 ?- jug(2,1).

false.

[2] 10 ?- jug(2,0).

Task Completed

true.

4. Program checking a person eligible for voting.

Objective:

- Learn the different relational operators in prolog.
- Learn how to create a knowledge base.
- To perform different condition check.

Description:

How to check whether the person is eligible for voting??

Person's eligibility for voting is determined by using person's age and nationality.

If the person is resident of the India then only he/she can vote.

Second thing is person should be adult i.e. age should be 18+.

Following is the program to check whether the person is eligible for voting or not.

Facts and Rules:

```
voter(A,B) :- ( A > 17 , B == 'ind' ),
```

```
nl,nl, write(' Voter is eligble ').
```

```
voter(A,B) :- ( A < 18 ; B \== 'ind ' ),
```

```
nl,nl, write('Voter is not eligible ').
```

Errors:

```
1 ?-
```

```
% c:/users/gokul/desktop/ai/voting compiled 0.02 sec, 3 clauses
```

```
ERROR: c:/users/gokul/desktop/ai/voting.pl:2: Syntax error: Operator expected
```

```
% c:/users/gokul/desktop/ai/voting compiled 0.03 sec, 58 clauses
```

```
ERROR: c:/users/gokul/desktop/ai/voting.pl:4:
```

```
Full stop in clause-body? Cannot redefine ./2
```

```
% c:/users/gokul/desktop/ai/voting compiled 0.02 sec, 2 clauses
```

```
% c:/users/gokul/desktop/ai/voting compiled 0.00 sec, 3 clauses
```

Results and Conclusion:

```
1 ?- voter(18,ind).
```

Voter is eligible
true .
2 ?- voter(18,pak).
Voter is not eligible
true.
3 ?- voter(17,ind).
Voter is not eligible
true .
4 ?- voter(17,pak).
Voter is not eligible
true .
5 ?- voter(0,ind).
Voter is not eligible
true .
6 ?- voter(0,0).
Voter is not eligible
true .
7 ?- voter(100,ind).
Voter is eligible
true .
8 ?- voter(19,oo).
Voter is not eligible
true.
9 ?- voter(18,ind).
Voter is eligible
true .

5. Program to calculate factorial of a number

Objective:

- How to create knowledge base
- How to do mathematical operations in prolog

Description:

The following is a mathematical definition of the factorial function:

$$factorial(N) = \begin{cases} 1 & N = 0 \\ N * factorial(N - 1) & N > 0 \end{cases}$$

This code abounds with special things to remember. The commas outside of parentheses mean and. The ':'- means if. The keyword 'is' is assignment, but "N is N-1" has no meaning in Prolog- there is no destructive assignment. There are no functions in Prolog, only relations. When we write 'factorial (N, M)' we are saying "N is related to M by the relation 'factorial'." The order is not important provided it is used consistently. The assertions about the relation 'factorial' end with a period. Variable names begin with upper-case letters, constants are numbers or names beginning with a lower-case letter. We know the formula for calculating n! (factorial of n) is:

$$n! = n * (n-1)!$$

We can interpret this simple mathematical equation into a Prolog program. To do so, we must determine the basis of the recursion, $0! = 1$ We will use two predicates here,

- Factorial predicate with one argument N, that will calculate and N!
- Factorial predicate with two arguments N and X. This function is recursively used. It will also calculate N!, but store it in the argument X in the process if recursion.

Facts and Rules:

```
fact(X,Y):- X is
0, Y is 1;
X>0,
N is X-1,
fact(N,G),
Y is X*G.
```

Errors:

```
1 ?-
% d:/prolog/prolog programs/prac 5/fact compiled 0.00 sec, 4 clauses
```

```
1 ?- factorial(5).
ERROR: Undefined procedure: factorial/1
ERROR: However, there are definitions for:
ERROR: factorial/2
false.
```

```
2 ?- factorial(X,10).
ERROR: factorial/2: Arguments are not sufficiently instantiated
```

Result and Conclusion:

```
1 ?- fact(0,Y).
Y = 1 .
```

```
2 ?- fact(1,Y).
Y = 1 .
```

```
3 ?- fact(2,Y).
Y = 2 .
```

```
4 ?- fact(3,Y).
Y = 6 .
```

```
5 ?- fact(4,Y).
Y = 24 .
```

```
6 ?- fact(5,Y).
Y = 120 .
```

7 ?- fact(6,Y).
Y = 720 .

8 ?- fact(7,Y).
Y = 5040 .

9 ?- fact(8,Y).
Y = 40320 .

10 ?- fact(9,Y).
Y = 362880 .

11 ?- fact(10,Y).
Y = 3628800

12 ?- fact(11,Y).
Y = 39916800

13 ?- fact(12,Y).
Y = 479001600

14 ?- fact(13,Y).
Y = 6227020800

15 ?- fact(14,Y). Y
= 87178291200.

6. Program for generating Fibonacci series

Objectives:

- How to create a knowledge base in prolog.
- Learn to perform mathematical operations in prolog
- Learn recursive method invocation.

Description:

The Fibonacci numbers are the integer sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, ..., in which each item is formed by adding the previous two. The sequence can be defined recursively by

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Fibonacci number programs that implement this definition directly are often used as introductory examples of recursion. However, many other algorithms for calculating (or making use of) Fibonacci numbers also exist.

Facts and Rules:

```
fibonacci(0,1).
fibonacci(1,2).
fibonacci(X,Y):- X1 is X-1,
fibonacci(X1,Y1),
X2 is X-2,
fibonacci(X2,Y2),
Y is Y1+Y2,!
```

Error:

```
1 ?-
% d:/prolog/prolog programs/prac 6/fibonacci compiled 0.00 sec, 4
clauses 1 ?- fibonacci(10).
```

ERROR: Undefined procedure: fibonacci/1
ERROR: However, there are definitions for:
ERROR: fibonacci/2
false.

2 ?- fibonacci(X,9).

ERROR: fibonacci/2: Arguments are not sufficiently instantiated

Result and Conclusion:

1 ?- fibonacci(4,P).

P = 8.

2 ?- fibonacci(6,P).

P = 21.

3 ?- fibonacci(1,P).

P = 2 .

4 ?- fibonacci(3,P).

P = 5.

5 ?- fibonacci(7,P).

P = 34.

6 ?- fibonacci(2,P).

P = 3.

7 ?- fibonacci(5,P).

P = 13.

8 ?- fibonacci(8,P).

P = 55.

9 ?- fibonacci(9,P).

P = 89.

10 ?- fibonacci(10,P).

P = 144.

11 ?- fibonacci(11,P).

P = 233.

12 ?- fibonacci(12,P).

P = 377

13 ?- fibonacci(13,P). P = 610.

14 ?- fibonacci(14,P). P = 987.

15 ?- fibonacci(15,P). P = 1597

7. Program for generating pyramid

Objective:

- How to create a knowledge base
- Learn to generate a pyramid using prolog

Description:

Pyramid is figure look like a triangle. As we know the pyramid in Ijpt country.

So we have to create a pyramid here in 1D space which looks exactly like a triangle.



Following is the program that create figure like pyramid using ‘*’ symbol only.

Facts and Rules:

```
pmy(X):- Y is 1,check1(X,Y).
```

```
check1(X,Y):- X>0 ->(space(X),star(Y),
```

```
    A is X-1,B is Y+2,
```



```
check1(A,B));(write(' ')).
```

```
space(Z):- Z>0 ->( write(' '),  
            K is Z-1,  
            space(K)); (Z is 0 ).
```

```
star(C):- C>0 ->(write('*'), M  
              is C-1,  
              star(M));(nl).
```

Errors:

```
ERROR: c:/users/gokul/documents/prolog/pyramid.pl:2: Syntax error: Operator  
expected % c:/users/gokul/documents/prolog/pyramid compiled 0.03 sec, 59 clauses
```

```
ERROR: c:/users/gokul/documents/prolog/pyramid.pl:5: Syntax error: Operator  
expected % c:/users/gokul/documents/prolog/pyramid compiled 0.00 sec, 3 clauses
```

```
ERROR: c:/users/gokul/documents/prolog/pyramid.pl:5: Syntax error: Operator expected
```

```
ERROR: c:/users/gokul/documents/prolog/pyramid.pl:7: Syntax error: Operator expected
```

```
% c:/users/gokul/documents/prolog/pyramid compiled 0.00 sec, 3 clauses
```

```
% c:/users/gokul/documents/prolog/pyramid compiled 0.00 sec, 5 clauses
```

Results and Conclusion:

```
1 ?- pmy(0).
```

```
true.
```

```
2 ?- pmy(5).
```

```
 *
***
*****
*****
*****
```

true.

3 ?- pmy(10).

```
 *
***
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

true.

4 ?- pmy(15).

```
 *
***
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

true.

5 ?- pmy(20).

8. Program for Towers of Hanoi puzzle

Objective:

- How to use prolog language
- How to create knowledge base
- How to analyze problem using divide and conquer methodology
- How to made recursive call in prolog

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.

The object of the puzzle is to move all the disks over to the right pole, one at a time, so that they end up in the original order on that pole. You can use the middle pole as a temporary resting place for disks, but at no time is a larger disk to be on top of a smaller one. It's easy to solve the Towers of Hanoi with two or three disks, but the process becomes more difficult with four or more disks.

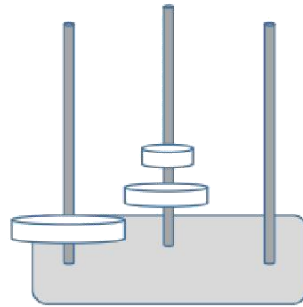
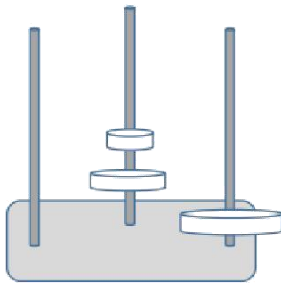
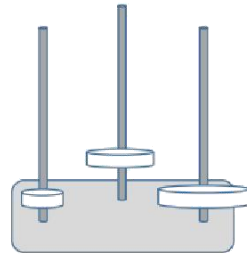
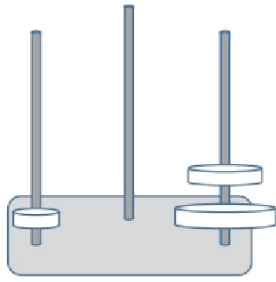
A simple strategy for solving the puzzle is as follows:

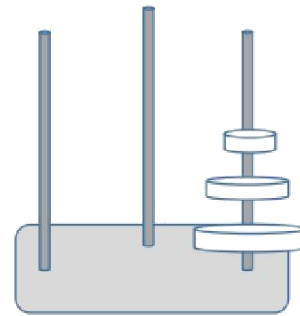
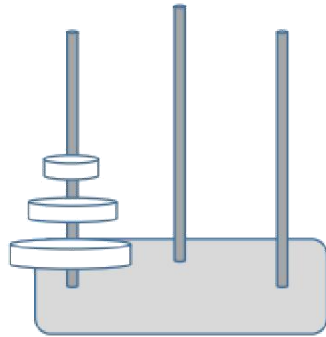
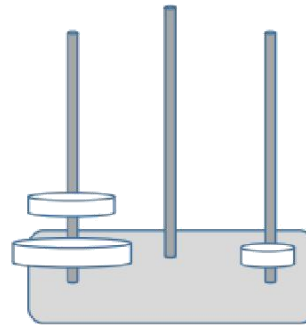
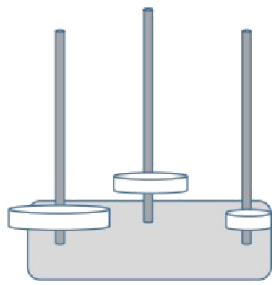
- You can move a single disk directly.
- You can move N disks in three general

steps:

- Move N-1 disks to the middle pole.
- Move the last (Nth) disk directly over to the right pole.
- Move the N-1 disks from the middle pole to the right pole.

The Visual Prolog program to solve the Towers Of Hanoi puzzle uses three predicates: Hanoi, with one parameter that indicates the total number of disks you are working with move, which describes the moving of N disks from one pole to another--using the remaining pole as a temporary resting place for disks inform, which displays what has happened to a particular disk.





Facts and Rules:

```
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).
```

Errors:

1 ?-

% d:/prolog/prolog programs/prac8/tower_of_hanoi compiled 0.00 sec, 3 clauses

1 ?- move(3,a,b).

ERROR: Undefined procedure: move/3

ERROR: However, there are definitions for:

ERROR: move/5

false.

2 ?- move(4,a,b,0).

ERROR: Undefined procedure: move/4

ERROR: However, there are definitions for:

ERROR: move/5

false.

Results and Conclusion:

?-

move(3,left,right,center).

Move top disk from left to right

Move top disk from left to center

Move top disk from right to center

Move top disk from left to right

Move top disk from center to left

Move top disk from center to right

Move top disk from left to right

true .

?-

move(2,left,right,center).

Move top disk from left to center

Move top disk from left to right

Move top disk from center to right

true

?-

move(4,left,right,center).

Move top disk from left to center

Move top disk from left to right

Move top disk from center to right

Move top disk from left to center

Move top disk from right to left

Move top disk from right to center

Move top disk from left to center

Move top disk from left to right

Move top disk from center to right

Move top disk from center to left
Move top disk from right to left
Move top disk from center to right
Move top disk from left to center
Move top disk from left to right
Move top disk from center to right
true

4 ?- move(5,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from right to left
Move top disk from center to left
Move top disk from right to center
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
true

9. Design an expert system

Objective:

- How to use prolog to create expert system
- Expert system on animal classification
- Apply complete prolog language constructs to design expert system

Description:

In artificial intelligence, an expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning about knowledge, represented primarily as if-then rules rather than through conventional procedural code. The first expert systems were created in the 1970s and then proliferated in the 1980s. Expert systems were among the first truly successful forms of AI software.

An expert system is divided into two sub-systems: the inference engine and the knowledge base. The knowledge base represents facts and rules. The inference engine applies the rules to the known facts to deduce new facts. Inference engines can also include explanation and debugging capabilities

Facts and Rules:

animal(tiger).

animal(lion).

animal(horse).

animal(dog).

animal(cat).

animal(ox).

animal(cow).

veg(cow).
veg(ox).
veg(horse).
nonveg(tiger).
nonveg(lion).
vegnonveg(dog).
vegnonveg(cat).
pet(cow).
pet(ox).
pet(cat).
pet(dog).
pet(horse).
wild(tiger).
wild(lion).
male(tiger).
male(lion).
male(horse).
male(dog).
male(ox).
female(cat).
female(cow).

check(X):- animal(X)->
 (nl,write('Animal'), check1(X));

```
write('no information available').
```

```
check1(X):-check2(X),veg(X)->(nl,write(X),write(' is veg')); nonveg(X)->(nl,write(X),write(' is non veg')); ( nl,write(X), write(' is veg and nonveg')).
```

```
check2(X):-check3(X), pet(X)->(nl,write(X),write(' is pet')); (nl,write(X),write(' is wild')).
```

```
check3(X):- nl,write(X),write(' has 4 leg'), nl,write(X), write(' has tail'), nl,write(X), check4(X).
```

```
check4(X):-male(X)->write(' is male'); write(' is female').
```

Errors:

1 ?-

Warning: c:/users/gokul/documents/prolog/expert.pl:34: Clauses of female/1 are not together in the source-file

% c:/users/gokul/documents/prolog/expert compiled 0.02 sec, 91 clauses

ERROR: c:/users/gokul/documents/prolog/expert.pl:9: Syntax error: Unexpected end of clause % c:/users/gokul/documents/prolog/expert compiled 0.00 sec, 33 clauses

ERROR: c:/users/gokul/documents/prolog/expert.pl:10: Syntax error: Operator expected

16c:/users/gokul/documents/prolog/expert compiled 0.00 sec, 31

clauses Warning: c:/users/gokul/documents/prolog/expert.pl:10:

Singleton variables: [Cow]

17c:/users/gokul/documents/prolog/expert compiled 0.02 sec, 34 clauses

ERROR: c:/users/gokul/documents/prolog/expert.pl:13: Syntax error: Operator
expected % c:/users/gokul/documents/prolog/expert compiled 0.03 sec, 32 clauses

Results and Conclusion:

1 ?- check(tiger).

Animal

tiger has 4 leg

tiger has tail

tiger is male

tiger is wild

tiger is non

veg true.

2 ?- check(koyal).

no information available

true.

3 ?- check(cow).

Animal

cow has 4 leg

cow has tail cow
is female cow is
pet cow is veg
true.

4 ?- check(demo).

no information available
true.

5 ?- check(human).

no information available
true.

6 ?- check(ox).

Animal

ox has 4 leg ox
has tail ox is
male ox is pet
ox is veg
true.

7 ?- check(dog).

Animal

dog has 4 leg
dog has tail dog
is male
dog is pet
dog is veg and nonveg
true.

8 ?- check(cat).

Animal

cat has 4 leg cat

has tail cat is

female cat is pet

cat is veg and nonveg

true.

9 ?- check(horse).

Animal

horse has 4 leg

horse has tail

horse is male

horse is pet horse

is veg true.

10 ?- check(parrot).

no information available

true.