LIST OF EXPERIMENTS

Sr No.	Name of Experiments
1	Study of Prolog.
2	Write a program to implement union and intersection operations on two lists.
3	Write a program to solve water jug problem.
4	Write a program to indicate disease by knowing symptom.
5	Write a program to implement monkey banana problem.
6	Write a program to implement traveling salesman problem.
7	Write a program to implement tower of Hanoi.
8	Write a program to implement path finder.
9	Write a Program to implement Depth First Search.
10	Write a program to solve 8 puzzles problem with Best First search.
11	Write a program to solve 8 Queen Problem.
12	Write a program to solve ROBOT traversal problem using Means End Analysis.

EXPERIMENT NO. 3

OBJECTIVE: Write a program to solve water jug problem.

Algorithm:

Problem: Given two jugs, a 4-gallon and a 3-gallon. Neither has any measuring marker on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?

Step1: Take two Jug A and Jug B of Capacity 4 gallon and 3 gallon respectively.

Step2: Goal is to achieve Jug A with Capacity of 2 gallon water and Jug B will be Empty , i.e. 0 gallon.

Step3: Represent the problem for machine by defining the rules as:

1. Fill the jugs with water of their capacity.
2. Empty jug A and fill it with 3 gallon water from jug B.
3. Fill Jug B again with 3 gallon water.
4. Fill Jug A completely by jug B.
5. Now in Jug a there is 4 gallon water and Jug B with 2 gallon water.
6. Empty Jug A and fill it with 2 gallon water from Jug B.
7. Goal Achieved.

Source Code:

```
Domains
        A,B=integer
Predicates
        go
        water(A,B)
clauses
        go:-
        write("enter capacity of jug A"),
        readint(A),nl,
        A<=4,
        write("enter capacity of jug B"),
        readint(B),nl,
        B<=3,
        write(A," ",B),nl,
        water(A,B).
        water(2,0):-
        write("goal achieved"),nl.
        water(2,_):-
        write("2_0"),nl,
        water(2,0).
        water(0,2):-
        write("2_0"),nl,
        water(2,0).
        water(_,2):-
```

```
write("0_2"),nl,

water(0,2).

water(4,2):-

write("0_2"),nl,

water(0,2).

water(3,3):-

write("4_2"),nl,

water(4,2).

water(3,0):-

write("3_3"),nl,

water(3,3).

water(0,3):-

write("3_0"),nl,

water(3,0).

water(4,3):-

write("0_3"),nl,

water(0,3).
```

Test data:

go

enter capacity for Jug A 3

enter capacity for Jug B 3

Output:

EXPERIMENT NO. 4

OBJECTIVE: Write a program to indicate disease by knowing symptoms using objects.

Algorithm:

Step1: Create two symbols to represent the disease name and indication of the disease.

Step2: Specify rules to create knowledge base to identify which disease can be possible for a list of symptoms/indications.

Source Code:

```
domains
        disease,indication=symbol
predicates
        symptom(disease,indication)
clauses
        symptom(chicken_Pox,high_fever).
        symptom(chicken_pox,chills).
        symptom(flu,running_nose).
        symptom(flu,chills).
        symptom(cold,body_ache).
        symptom(cold,high_fever).
        symptom(cold,running_nose).
        symptom(flu,moderate_cough).
```

Test data:

Symptom(flu,chills)

Symptom(cold,chills)

Output:

EXPERIMENT NO. 5

OBJECTIVE: Write a program to implement Monkey Banana Problem.

Algorithm:

Monkey Banana Problem: How a monkey can get Bananas from a banana tree if he has alternatives like Height of Tree, Stick, Table, and his Jump?

Step1: Create the for integers to represent his Height, Stick, Table, and his Jump.

Step2: Generate the possible cases of his attempts to get the bananas.

1. If Height of tree<= Monkey’s Jump then Successful.
2. If Height of Tree<=Jump+Stick then Successful.
3. If Height of Tree<=Jump+Table then Successful.
4. If Height of Tree<=Jump+Stick+Table then Successful.
5. If Height of Tree>Jump+Stick+Table then Unsuccessful.

Source Code:

```
domains
        Height,Jump,Stick,Table=integer
predicates
        go
        attempt(integer,integer,integer,integer,integer)
        task(integer)
clauses
        go:-
        write("1.monkey jump"),nl,
        readint(Jump),
        write("2.Stick Length"),nl,
        readint(Stick),
        write("3.Height of ceiling"),nl,
        readint(Height),
        write("4.Height of table"),nl,
        readint(Table),
        attempt(_,Height,Jump,Stick,Table),
        readint(_).
attempt(0,Height,Jump,_,_):-
        write("monkey jumped to catch banana"),nl,
        Height<=Jump,
        task(1).
attempt(1,Height,Jump,Stick,_):-
        write("monkey jumped with stick to catch banana"),nl,
        Height<=Jump+Stick,
        task(1).
attempt(2,Height,Jump,Stick,Table):-
        write("monkey jumped with table having stick to catch banana"),nl,
        Height<=Jump+Stick+Table,
        task(1).
attempt(3,Height,Jump,_,Table):-
        write("monkey jumped with table to catch banana"),nl,
        Height<=Jump+Table,
        task(1).
attempt(4,Height,Jump,Stick,Table):-
        write("monkey jumped with table having stick to catch banana"),nl,
        Height>=Jump+Table+Stick,
        write("unsuccessful").
task(1):-
        write("successful").
```

Test data:

go
1. monkey Jump
30
2. Stick Length
20
3. Height of ceiling
50
4. Height of Table
10

Output:

EXPERIMENT NO. 6

OBJECTIVE: Write a program to implement Travelling Salesman Problem.

Algorithm:

Travelling Salesman Problem: A salesperson must visit n cities, passing through each city only once, beginning from one of the city that is considered as a base or starting city and returns to it. The cost of the transportation among the cities is given. The problem is to find the order of minimum cost route that is, the order of visiting the cities in such a way that the cost is the minimum.

Step1: Create a complete route from one city to other city.

Step2: First, find out all (*n* -1)! Possible solutions, where *n* is the number of cities.

Step3: Next, determine the minimum cost by finding out the cost of everyone of these (*n* -1)! Solutions.

Step4: Finally, keep the one with the minimum cost.

Source Code:

```
domains
        A,B,C,D=symbol
predicates
        go
        route(symbol,symbol,integer)
        direct(symbol,symbol,integer)
        indirect(symbol,symbol,integer)
clauses
        route(a,a,0).
        route(a,b,5).
        route(a,c,6).
        route(b,b,0).
        route(c,c,0).
        route(a,d,7).
        route(d,d,0).
        route(b,d,2).
        route(c,d,1).
go:-
        write("enter first city"),nl,
        readln(A),
        write("enter destination city"),nl,
        readln(B),
        direct(A,B,X),
        indirect(A,B,Y),
        X>Y,
        write("choose indirect path"),nl,
        Y>X,
        write("choose direct path"),nl.
       direct(A,B,X):-
        route(A,B,X),
        write("successfull"),nl.
        indirect(A,B,Y):-
        route(A,R,T),
        route(R,B,Z),
        write("successfull"),
        Y=T+Z.
```

Test data:

enter first city

d

enter destination city

a

Output:

EXPERIMENT NO. 7

OBJECTIVE: Write a program to implement Towers of Hanoi Problem.

Algorithm:

The objective 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.

Step1: Take N to represent total number of disks and three peg as A, B, and C.

Step2: Check if N=1 Then Move disk from source to destination.

Step3: Else move J disks (J=N-1) from source to destination, then move disc N from A to C and finally

move N−1 discs from B to C so they sit on disc *N*.

Source Code:

```
domains

        N,J=integer

        FROM,TEMP,TO=char

predicates

        go

        transfer(integer,char,char,char)

clauses

        go:-

        write("enter no of disks"),nl,

        readint(N),

        transfer(N,'A','B','C').

        transfer(N,FROM,TO,TEMP):-

        N>0,

        J=N-1,

        transfer(J,FROM,TEMP,TO),

        write("add disk",J,"from",FROM,"to",TO),nl,

        readchar(_),

        transfer(J,TEMP,TO,FROM),

        true.

        transfer(N,FROM,TEMP,TO):-

        true.
```

Test data:

go

enter no of disks

3

Output: