ABES Institute of Technology
Ghaziabad

Affiliated to Dr. A.P.J. AKTU, Lucknow

LAB FILE

Department of Computer Science and Engineering
Subject Name : Artificial Intelligence Lab
Subject Code : RCS-752
Session : 2020 - 2021
Semester : 7th

Submitted to : Ms. Aishwarya Gupta

Submitted by: Sandeep Kumar Shukla
Roll No. : 1729010140
Section : 4CSE - C

Index

S No.	Program
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 LISP
10.	Write a Lisp program to solve best first search traversal.

ABESIT
COLLEGE OF ENGINEERING

ABES Institute of Technology, Ghaziabad
Department of Computer Science & Engineering

Institute Vision	**To be leading institution in technical education providing education and training enabling human resource to serve nation and world at par with global standards in education**
Institute Mission	**1) Developing state of art infrastructure which also includes establishment of center of excellence in pursuit of academic and technical excellence** **2) Valuing work force inculcating belongingness and professional integrity** **3) To develop human resource to solve local, regional and global problems to make technology relevant to those who mean it most**

ABESIT
COLLEGE OF ENGINEERING

ABES Institute of Technology, Ghaziabad
Department of Computer Science & Engineering

Department Vision	**To provide excellence by imparting knowledge to the learners enabling them to become skilled professionals to be recognized as a responsible citizen.**
Department Mission	**1) Provide quality education in the field of computer science and engineering through experienced and qualified faculty members.** **2) Motivate learners for higher studies and research oriented activities by utilizing resources of Centers of Excellence.** **3) Inculcate societal values, professional ethics, team work, and leadership qualities by having exposure at National and International level activities.**

ABES Institute of Technology, Ghaziabad
Department of Computer Science & Engineering

Program Educational Objective (PEOs)

POE 1	**Graduates of the program are expected to be employed in IT industry or Indulge in higher studies and research.**
POE 2	**Graduates of the program are expected to exhibit curiosity to learn new technologies and work with ethical values and team work.**
POE 3	**Graduates of the program are expected to design and develop innovative solutions related to real world problems of the society.**

Program Specific Outcome (PSOs)

PSO 1	**Solve complex problems using data structures and other advanced suitable algorithms.**
PSO 2	**Interpret fundamental concepts of computer systems and understand its hardware and software aspect.**
PSO 3	**Analyze the constraints of the existing data base management systems and get experience on large-scale analytical methods in the evolving technologies.**
PSO 4	**Develop intelligent systems and implement solutions to cater the business specific requirements.**

List of Programs

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 LISP**
10. **Write a Lisp program to solve best first search traversal.**

PROGRAM 1
STUDY OF PROLOG

1.1 Study of Prolog

The Prolog language allows us to explore a wide range of topics in discrete mathematics, logic, and computability. Prolog's powerful pattern-matching ability and its computation rule give us the ability to experiment in two directions. For example, a typical experiment might require a test of a definition with a few example computations Prolog computation rule also allows a definition to be tested in reverse, by specifying a result and then asking for the elements that give the result.

Following are feature of prolog programming language

1. In purely declarative languages, the programmer only states what the problem is and leaves the rest to the language system.
2. We'll see specific, simple examples of cases where Prolog fits really well shortly.
3. Prolog = Programmation en Logique (Programming in Logic).
4. In a declarative language the programmer specifies a goal to be achieved the Prolog system works out how to achieve it
5. Relational databases owe something to Prolog
6. Prolog (programming in logic) is a logic-based programming language: programs correspond to sets of logical formulas and the Prolog interpreter uses logical methods to resolve queries.
7. Prolog is a declarative language: you specify what problem you want to solve rather than how to solve it.
8. Prolog is very useful in some problem areas, such as artificial intelligence, natural language processing, databases, but pretty useless in others, such as for instance graphics

Application of Prolog

1. Intelligent data base retrieval
2. Natural language understanding
3. Expert systems
4. Specification language
5. Machine learning
6. Robot planning
7. Automated reasoning
8. Problem solving

To start the Prolog interpreter in a UBANTO environment type and hit return. Once Prolog has started up it displays the prompt

```
|?-
```

which indicates that the interpreter is waiting for a command from the user. All commands must end with a period. For example, the command

```
|?- integer(3.4).
```

returns the answer no because 3.4 is not an integer. A command is usually called a ***goal or a query***. A Prolog program is a set of facts or rules called ***definite clauses***.

Loading Information

To read in the contents of a file named ***filename*** type

```
|?- [filename]
```

and hit return. If the file name contains characters other than letters or digits, then put single quotes around the filename. For example, if the name of the file is ***file . p,*** then type

```
|?- ['file.p'].
```

will load the file named file.p.
You can read in several files at once. For example, to read in files named foo, goo, and moo type

```
|?- [foo, goo, moo].
```

Sometimes it may be useful to enter a few clauses directly from the a terminal to test something or other. In this case you must type the command

```
|?- [user].
```

and hit return. The prompt will appear to indicate that the interpreter is waiting for data.

Variables, Predicates, and Clauses

Variable

A variable may be represented by a string of characters made up of letters or digits or the underscore symbol _, and that begins with either an uppercase letter or _. For example, the following strings denote variables:
X, Input_list, Answer, _,
A variable name that begins with the underscore symbol represents an unspecified (or anonymous) variable.

Predicates

A predicate is a relation. In Prolog the name of a predicate is an alphanumeric string of characters (including _) that begins with a lowercase letter.

Clauses

The power of Prolog comes from its ability to process clauses. A clause is a statement taking one of the forms head.or head :- body. where head is an atomic formula and body is a sequence of atomic formulas separated by commas. For example, the following statements are clauses.

```
capital–of(salem, oregon).
q(b).
p(X) :- q(X), r(X), s(X).
```

The last clause has head p(X) and its body consists of the atomic formulas

```
q(X), r(X), and s(X).
```

The meaning of a clause of the form head is that head is true. A clause of the form

```
head :- body.
```

has a declarative meaning and a procedural meaning. The declarative meaning is that the head is true if all of the atomic formulas in the body are true. The procedural meaning is that for head to succeed, each atomic formula in the body must succeed. For example, suppose we have the clause

```
p(X) :- q(X), r(X), s(X).
```

From the declarative point of view this means that for all X, p(X) is true if q(X) and r(X) and s(X) are true. From the procedural point of view this means that for all X, p(X) succeeds if q(X), r(X), and s(X) succeed.

Or Clauses

Prolog also allows us to represent the “or” operation in two different ways. For example, suppose that we have the following two Prolog clauses to define the parentof relation in terms of the mother of and father of relations.

```
parentof(X, Y) :- motherof(X, Y).
parentof(X, Y) :- fatherof(X, Y).
```

Unification

Unification is the process of matching two expressions by attempting to construct a set of bindings for the variables so that when the bindings are applied to the two expressions, they become syntactically equal. Unification is used as part of Prolog’s computation rule. The following symbol is used for unification within a program. If two expressions can be unified, then Prolog will return with corresponding bindings for any variables that occur in the expressions. For example, try out the following tests.

```
|?- b = b.
|?- p(a) = p(a).
|?- p(X) = p(b).
|?- 5 = 5.
|?- 2 + 3 = 1 + 4.
```

Computation

A Prolog program executes goals, where a goal is the body of a clause. In other words, a goal is one or more atomic formulas separated by commas. The atomic formulas in a goal are called subgoals. For example, the following expression is a goal consisting of two subgoals.

```
|?- par(X, james), par(Y, X).
```

The execution of a goal proceeds by unifying the subgoals with heads of clauses. The search for a matching head starts by examining clauses at the beginning of the program and proceeds linearly through the clauses. If there are two or more subgoals, then they are executed from left to right. A subgoal is true in two cases:
1. It matches a fact (i.e., the head of a bodyless clause).
2. It matches the head of a clause with a body and when the matching substitution is applied to the body, each subgoal of the body is true. A goal is true if there is a substitution that when applied to its subgoals makes each subgoal true.

For example, suppose we have the following goal for the introductory program example.
|?- par(X, james), par(Y, X).
This goal is true because there is a substitution {X=ruth, Y=katherine} that when applied to the two subgoals gives the following subgoals, both of which are true. par(ruth, james), par(katherine, ruth).

Experiments to Perform

1. Try out some unification experiments like the following. First find the answers by hand. Then check your answers with Prolog.

```
|?- p(X) = p(a).
|?- p(X, f(Y)) = p(a, Z).
|?- p(X, a, Y) = p(b, Y, Z).
|?- p(X, f(Y, a), Y) = p(f(a, b), V, Z).
|?- p(f(X, g(Y)), Y) = p(f(g(a), Z), b).
```

Numeric Computations

Prolog has a built-in predicate “is” that is used to evaluate numerical expressions. The predicate is infix with a variable on the left and a numerical expression on the right. For example, try out the following goals.

```
|?- X is 5 + 7.
|?- X is 5 - 4 + 2.
|?- X is 5 * 45.
|?- X is log(2.7).
|?- X is exp(1).
|?- X is 12 mod 5.
```

The expression on the right must be able to be evaluated. For example, try out the goal

```
|?- X is Y + 1.
```

Now try out the goal

```
|?- Y is 5, X is Y + 1.
```

SICStus Prolog has a rich set of numerical operations that includes all of the ISO operations: Binary operators +, –, *, /, //, rem, mod, sin, cos, atan. Unary operators +, –, abs, ceiling, floor, float, truncate, round, exp, sqrt, log.

Numeric Comparison

Numerical expressions can be compared with binary comparison operators. The six operators are given as follows:

```
=:=,  =\=, <,>,=<,>=.
```

For example, try out the following goals.

```
|?- 5 &amp;lt; 6 + 2.
|?- 5 =&amp;lt; 6 + 2.
|?- 5 =:= 6 –
```

Family Trees

In this experiment we’ll continue working with a family tree by examining a few of the many family relations. To keep things short and concise, let p(X, Y) mean that X is a parent of Y, and let g(X, Y) mean that X is a grandparent of Y.

Outcome-Students study logic programming language Prolog.

PROGRAM 2

Write simple fact for the statement using Prolog

1. Enter the following program into a file.

```
p(a, b).
p(a, c).
p(a, d).
p(b, e).
p(b, f).
p(c, g).
p(d, h).
p(e, i).
p(g, j).
p(h, k).
g(X, Y) :- p(X, Z), p(Z, Y).
```

i. Facts about a hypothetical computer science department:

```
% lectures(X, Y): person X lectures in course Y
    lectures( turing , 9020).
    lectures(codd, 9311).
    lectures(backus, 9021).
    lectures(ritchie, 9201).
    lectures(minsky, 9414).
    lectures(codd, 9314).
% studies(X, Y): person X studies in course Y
    studies(fred, 9020).
    studies(jack, 9311).
    studies(jill, 9314).
    studies(jill, 9414).
    studies(henry, 9414).
    studies(henry, 9314).
%year(X, Y): person X is in year Y
    year(fred, 1).
    year(jack, 2).
    year(jill, 2).
    year(henry, 4).
```

Together, these facts form Prolog's database.
To ask Prolog to find the course that Turing teaches, enter this:

```
?- lectures(turing, Course).
Course = 9020  ← output from Prolog.
```

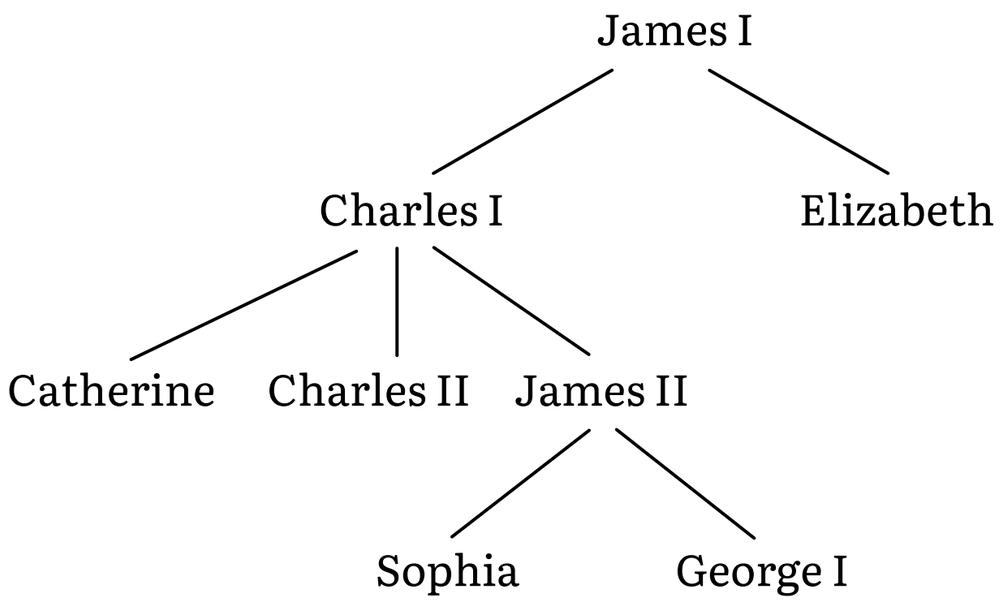
2. Here are some simple clauses.

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).
```

The following queries yield the specified answers.

```
|?- likes(mary,food).
yes.
|?- likes(john,wine).
yes.
|?- likes(john,food).
no.
```

3. Slightly more complicated family tree.



Here are the resultant clauses:

---

male(james1).
male(charles1).
male(charles2).
male(james2).
male(george1).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(george1, sophia).

Here is how you would formulate the following queries:

Was George I the parent of Charles I?

Query: parent(charles1, george1).

Who was Charles I's parent?

Query: parent(charles1,X).

Who were the children of Charles I?

Query: parent(X,charles1).

Outcome –Students learn basic Fact of prolog by doing program on family relation.

PROGRAM 3

Write predicates One converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing.

Arithmetic :
c_to_f ⟶ f is c * 9 / 5 +32
freezing ⟶ f < = 32

Rules:
c_to_f(C,F) :-
F is C * 9 / 5 + 32.
freezing(F) :-
F =< 32.

Output:
Queries:
?- c_to_f(100,X).
X = 212
Yes
?- freezing(15)
Yes
?- freezing(45).
No

Outcome : Student will understand how to write a program using the rules.

PROGRAM 4

Write a program to solve the Monkey Banana problem.

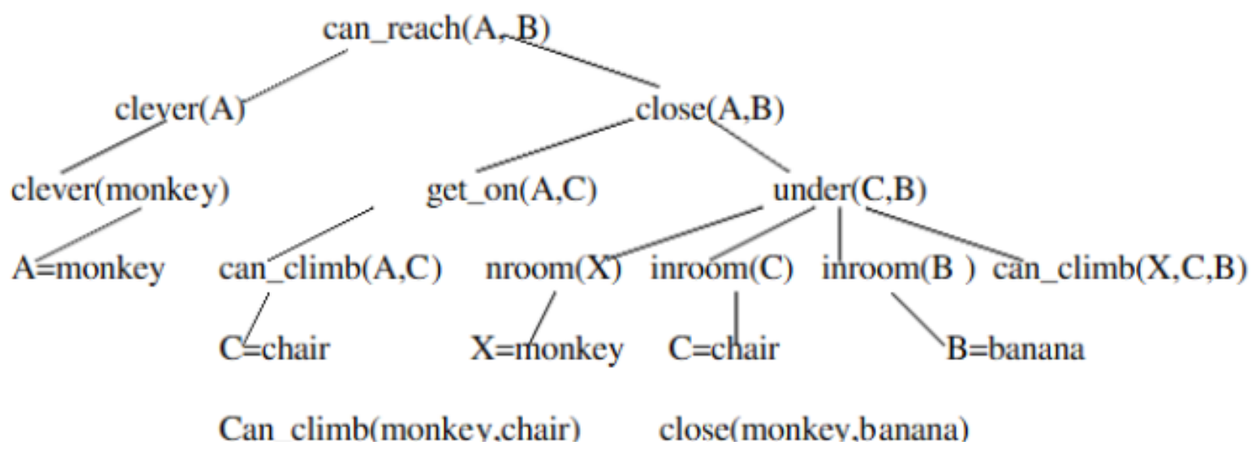
Imagine a room containing a monkey, chair and some bananas. That have been hanged from the centre of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair .The problem is to prove the monkey can reach the bananas.The monkey 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.

Production Rules

can_reach ⟶ clever,close.
get_on: ⟶ can_climb.
under ⟶ in room,in_room, in_room,can_climb.
Close ⟶ get_on,under| tall

Parse Tree



Clauses:

```
in_room(bananas).
in_room(chair).
in_room(monkey).
clever(monkey).
can_climb(monkey, chair).
tall(chair).
can_move(monkey, chair, bananas).
can_reach(X, Y):-
clever(X),close(X, Y).
get_on(X,Y):- can_climb(X,Y).
under(Y,Z):-
in_room(X),in_room(Y),in_room(Z),can_climb(X,Y,Z).
close(X,Z):-get_on(X,Y),
under(Y,Z);
tall(Y).
```

Output:

Queries:

```
?- can_reach(A, B).
    A = monkey.
    B = banana.
?- can_reach(monkey, banana).
    Yes.
```

Outcome: Student will understand how to solve monkey banana problem using rules in prolog.

PROGRAM 5

Write program in turbo prolog for medical diagnosis and show the advantage and disadvantage of green and red cuts.

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

Outcome: Student will understand how to create a expert system using prolog

PROGRAM 6

Write a program to implement factorial, fibonacci of a given number.

Factorial:

```
factorial(0,1).

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

Output:

```
        Goal:
        ?- 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:

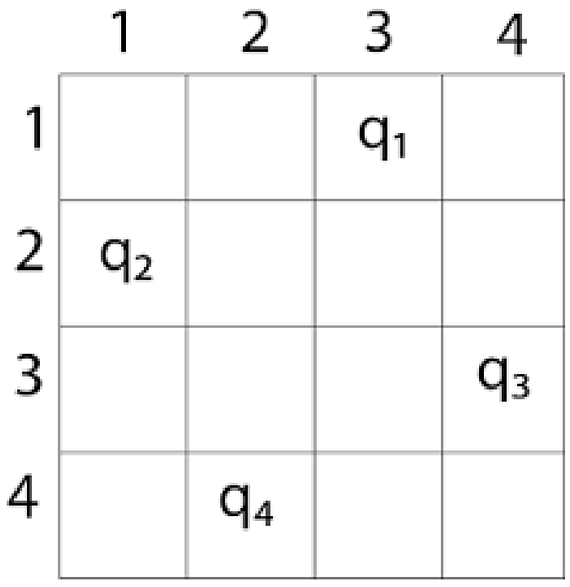
```
        Goal:
        ?-fib(10,X).
        X=55
```

Outcome: Student will understand the implementation of Fibonacci and factorial series using prolog.

PROGRAM 7

Write a program to solve 4-Queen problem.

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.




```
domains
        queen = q(integer, integer)
        queens = queen*
        freelist = integer*
        board = board(queens, freelist, freelist, freelist, freelist)

predicates
        nondeterm placeN(integer, board, board)
        nondeterm place_a_queen(integer, board, board)
        nondeterm nqueens(integer)
        nondeterm makelist(integer, freelist)
        nondeterm findandremove(integer, freelist, freelist)
        nextrow(integer, freelist, freelist)

clauses
        nqueens(N):-
        makelist(N,L),
        Diagonal=N*2-1,
        makelist(Diagonal,LL),
        placeN(N,board([],L,L,LL,LL),Final),
        write(Final).
        placeN(_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.
        placeN(N,Board1,Result):-
        place_a_queen(N,Board1,Board2),
        placeN(N,Board2,Result).
        place_a_queen(N,
        board(Queens,Rows,Columns,Diag1,Diag2),
        board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-
        nextrow(R,Rows,NewR),
        findandremove(C,Columns,NewC),
        D1=N+C-R,findandremove(D1,Diag1,NewD1),
        D2=R+C-1,findandremove(D2,Diag2,NewD2).
        findandremove(X,[X|Rest],Rest).
        findandremove(X,[Y|Rest],[Y|Tail]):-
        findandremove(X,Rest,Tail).
        makelist(1,[1]).
        makelist(N,[N|Rest]) :-
        N1=N-1,makelist(N1,Rest).
        nextrow(Row,[Row|Rest],Rest).
```

Output:

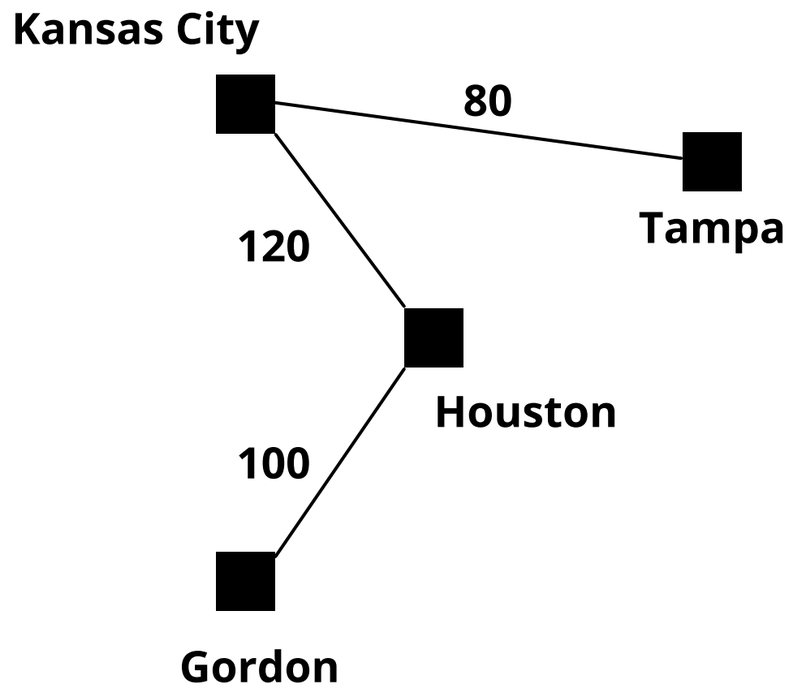
```
        Goal:
        ?-nqueens(4),nl.
            board([q(1,2),q(2,4),q(3,1),q(4,3),[],[],[7,4,1],[7,4,1])
            yes
```

Outcome : Student will implement 4-Queen problem using prolog

PROGRAM 8

Write a program to solve traveling salesman problem.

The following is the simplified map used for the prototype:



Production Rules:-

route(Town1,Town2,Distance)
road(Town1,Town2,Distance).
route(Town1,Town2,Distance)
road(Town1,X,Dist1),
route(X,Town2,Dist2),
Distance=Dist1+Dist2,

```
domains
        town = symbol
        distance = integer
        predicates
        nondeterm road(town,town,distance)
        nondeterm route(town,town,distance)

clauses
        road("tampa","houston",200).
        road("gordon","tampa",300).
        road("houston","gordon",100).
        road("houston","kansas_city",120).
        road("gordon","kansas_city",130).
        route(Town1,Town2,Distance):-
        road(Town1,Town2,Distance).
        route(Town1,Town2,Distance):-
        road(Town1,X,Dist1),
        route(X,Town2,Dist2),
        Distance=Dist1+Dist2,!.
```

Output:

Goal:
route("tampa", "kansas_city", X),
write("Distance from Tampa to Kansas City is ",X),nl.
Distance from Tampa to Kansas City is 320
X=320

Outcome : Student will implement travelling salesmen problem using prolog

PROGRAM 9

Write a program to solve water jug problem using LISP.

Program:

```
;returns the quantity in first jug
(defun get-first-jug (state) (car state))
;returns the quantity in second jug
(defun get-second-jug (state) (cadr state))
;returns the state of two jugs
(defun get-state (f s) (list f s))
;checks whether a given state is a goal
; GOAL IS TO GET 4 IN SECOND JUG
(defun is-goal (state)
 (eq (get-second-jug state) 4))
;returns all possible states that can be derived
;from a given state
(defun child-states (state)
 (remove-null
(list
 (fill-first-jug state)
 (fill-second-jug state)
 (pour-first-second state)
 (pour-second-first state)
 (empty-first-jug state)
 (empty-second-jug state))))
;remove the null states
(defun remove-null (x)
 (cond
((null x) nil)
((null (car x)) (remove-null (cdr x)))
((cons (car x) (remove-null (cdr x))))))
;return the state when the first jug is filled (first jug can hold 3)
(defun fill-first-jug (state)
(cond
((< (get-first-jug state) 3) (get-state 3 (get-second-jug state))))))
;returns the state when the second jug is filled (second jug can hold 5)
(defun fill-second-jug (state)
 (cond
((< (get-second-jug state) 5) (get-state (get-first-jug state) 5))))
;returns the state when quantity in first
;is poured to second jug
(defun pour-first-second (state)
 (let ( (f (get-first-jug state))
 (s (get-second-jug state)))
(cond
 ((zerop f) nil) ; first jug is empty
 ((= s 5) nil) ; Second jug is full
 ((<= (+ f s) 5)
 (get-state 0 (+ f s)))
 (t ; pour to first from second
 (get-state (- (+ f s) 5) 5)))))
;returns the state when second jug is poured to first
(defun pour-second-first (state)
 (let ( (f (get-first-jug state))
 (s (get-second-jug state)))
```

```
(cond
((zerop s) nil) ; second jug is empty
((= f 3) nil) ; second jug is full
((<= (+ f s) 3)
(get-state (+ f s) 0))
(t ;pour to second from first
(get-state 3 (- (+ f s) 3))))))
;returns the state when first jug is emptied
(defun empty-first-jug (state)
(cond
((> (get-first-jug state) 0) (get-state 0 (get-second-jug state)))))
;returns the state when second jug is emptied
(defun empty-second-jug (state)
        (cond
        ((> (get-second-jug state) 0) (get-state (get-first-jug state) 0))))


;;;MAIN FUNCTION
        (defun dfs (start-state depth lmt)
        (setf *node* 0)
        (setf *limit* lmt)
        (dfs-node start-state depth)
        )
        ;dfs-node expands a node and calls dfs-children to recurse on it
        (defun dfs-node (state depth)
         (setf *node* (+ 1 *node*))
         (cond
        ((is-goal state) (list state))
        ((zerop depth) nil)
        ((> *node* *limit*) nil)
        ((let ((goal-child (dfs-children (child-states state) (- depth 1))))
         (and goal-child (cons state goal-child)))) ;for back-tracking if the branch don't have a goal
state
        ))
        ;dfs-children expands each node recursively and give it
        ;to dfs-node to process
        (defun dfs-children (states depth)
         (cond
        ((null states) nil)
        ((dfs-node (car states) depth))
        ((dfs-children (cdr states) depth))))

        (print "ENTER YOUR INPUT AS")
        (print "(dfs start_state depth limit)")
```

Outcome : Student will implement water-jug problem using Lisp

PROGRAM 10

Write a Lisp program to solve best first search traversal.

Program

```
:;;;; Best First search
;;; The search net
        (setf (get 's 'neighbors) '(a d)
            (get 'a 'neighbors) '(s b d)
            (get 'b 'neighbors) '(a c e)
            (get 'c 'neighbors) '(b)
            (get 'd 'neighbors) '(s a e)
            (get 'e 'neighbors) '(b d f)
            (get 'f 'neighbors) '(e))
(setf (get 's 'coordinates) '(0 3)
      (get 'a 'coordinates) '(4 6)
      (get 'b 'coordinates) '(7 6)
       (get 'c 'coordinates) '(11 6)
       (get 'd 'coordinates) '(3 0)
       (get 'e 'coordinates) '(6 0)
       (get 'f 'coordinates) '(11 3))

(defun straight-line-distance (node1 node2)
"Computes the straight line distance between two nodes, whose position is specified by x and y
coordinates"
(let ((coordinate1 (get node1 'coordinates)) (
     coordinate2 (get node2 'coordinates)))   (
sqrt (+ (expt (- (first coordinate1) (first coordinate2)) 2)     (
     expt (- (second coordinate1) (second coordinate2)) 2)))))
(defun closerp (path1 path2 target-node)
"Return true if node1 (first path1) is close to target than node2, false otherwise"
(< (straight-line-distance (first path1) target-node)    (
     straight-line-distance (first path2) target-node)))
(defun extend-path (path)
"Extend the path by consing on a new neighbor node that does not result in loops"  (
mapcar #'(lambda (new-node) (cons new-node path))   (
     remove-if #'(lambda (neighbor) (member neighbor path)) (
     get (first path) 'neighbors))))
(defun best-first-search (start finish &optional (queue (list (list start))))  (
cond ((endp queue) nil)
       ((eq finish (first (first queue)))
       (reverse (first queue)))
       (t (print queue)
       (best-first-search start finish
       (sort
       (append (extend-path (first queue))
       (rest queue ))
      (lambda (p1 p2) (closerp p1 p2 finish)))))))

 #|
CL-USER> (best-first-search 's 'f)
((S))
((A S) (D S))
((B A S) (D A S) (D S))
((C B A S) (E B A S) (D A S) (D S))
((E B A S) (D A S) (D S))
(S A B E F)
CL-USER>
|#
```

```
;;; Problem 19-3
    (defun best-first-with-merge (start finish
                                  &optional (queue (list (list start))))
      "A more efficient version of best-first that does not sort previously sorted paths"
        (cond ((endp queue) nil)
              ((eq finish (first (first queue)))
              (reverse (first queue)))
              (t
              (print queue)
              (best-first-with-merge start  finish
              (merge 'list (sort (extend-path (first queue))
              #'(lambda (p1 p2) (closerp p1 p2 finish)))
                  (rest queue)
              #'(lambda (p1 p2) (closerp p1 p2 finish)))))))
    #|
    CL-USER> (best-first-with-merge 's 'f)

    ((S))
    ((A S) (D S))
    ((B A S) (D A S) (D S))
    ((C B A S) (E B A S) (D A S) (D S))
    ((E B A S) (D A S) (D S)) (S A B E F)
    CL-USER> |#
```

Outcome : Implement best-first-search traversal using Lisp