

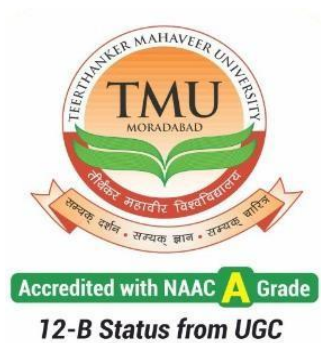
LAB MANUAL

ARTIFICIAL INTELLIGENCE LAB

(ECS-651)

VI Semester

FACULTY OF ENGINEERING



TEERTHANKER MAHAVEER UNIVERSITY

MORADABAD

2024



FACULTY OF ENGINEERING
(TEERTHANKERMAHAVEERUNIVERSITYMORADABAD)

LAB MANUAL

Submitted By

Student Name:.....

RollNo :.....

Branch:.....Section.....

Year Semester.....

Submitted To

FACULTY IN CHARGE

OBJECTIVES:

1. To introduce students to the basic knowledge of programming fundamentals of Prolog.
2. To impart writing skill of Prolog programming to the students and solving problems.
3. To impart the concepts like Fact, rules Variables Predicates.

COURSEOUTCOME:

Course Outcomes:	On completion of the course, the students will be :	BTL
CO1.	Understanding the role of PROLOG for implementation of solutions of AI problems	2
CO2.	Understanding the architecture and evaluation scheme of PROLOG	3
CO3.	Applying the PROLOG for solving trivial problems	3
CO4.	Monkey Banana problem using PROLOg	3
CO5.	Applying the various knowledge representation structures.	3

Program Outcomes for Engineering

PO-1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO-2	Problem analysis & Solving: Identify, formulate, research literature ,and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO-3	Design/development of solutions: Design solutions for complex engineering problems and design system components the public health and safety, and the cultural, societal, and environmental considerations or processes that meet the specified needs with appropriate consideration
PO-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.
PO-5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tool including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO-6	Social Interaction & effective citizenship: 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.
PO-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.
PO-8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO-9	Attitude (Individual and teamwork): Function effectively as an individual, and as member or leader in diverse teams, and in multidisciplinary settings.
PO-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 clean instructions.
PO-11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply the stone's own work ,as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO-12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes(PSOs)

PSO-1	Developing skills for creating computational solutions with emerging technologies, programming languages, mathematical foundations, algorithmic principles and open source platforms to solve complex engineering problems.	
PSO-2	Ability to understand the evolutionary changes in computing by applying standard practices and skills acquired through computer engineering to provide solutions with innovative ideas and interdisciplinary research.	
PSO-3	Ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur and pursue higher studies in their respective domains	

CO-PO Mapping

Cours e Code	CO		P O 1	PO2	PO3	P O 4	P O 5	P O 6	P O 7	PO8	P O 9	PO 1 0	PO 1 1	PO 1 2
ECS65 1	CO1	ECS651.1	2	1	1	1								
	CO2	ECS651.2	2	1	1	1	2							
	CO3	ECS651.3	3	2	2	1	3							
	CO4	ECS651.4	3	2	2	1	3							
	CO5	ECS651.5	2	1	1	1	2							
Mapping Strength		ECS651	2.4	1.4	1.4	1	3							

CO-PSO Mapping

Course Code	CO		PSO1	PSO2	PSO3
ECS651	CO1	ECS651.1	1	1	1
	CO2	ECS651.2	2	3	
	CO3	ECS651.3	1	2	
	CO4	ECS651.4	2	2	
	CO5	ECS651.5	2	2	2
Mapping Strength		ECS651	1.6	2	1.5

Lab Equipment

Following hardware and software are necessary to perform the experiments in prolog Programming Lab.

Hardware:

1. A computer that can execute Prolog programs.

Software:

1. Windows Operating System.
2. SWI Prolog

Programming Languages:

1. Prolog

Evaluation of practical course

Practical Courses Internal Evaluation(50marks)

Experiment (30Marks)	Attendance (10Marks)	Viva (10Marks)	Total Internal (50Marks)
-------------------------	-------------------------	-------------------	-----------------------------

Practical Courses External Evaluation(50marks)

The external evaluation would also be done by the External Examiner based on the experiment performed during the examination.

Experiment (30Marks)	File Work (10Marks)	Viva (10Marks)	Total External (50 Marks)
-------------------------	------------------------	-------------------	------------------------------

INDEX

Program: B.Tech-CSE

Semester - VI

Course: Artificial Intelligence Lab

Course Code: ECS 651

Lab Exercises

S. No.	Program No.	Page No.	Sign of faculty
1	1. Study of PROLOG Programming language and its Functions. Write simple facts for the statements using PROLOG.		
2	1. Write the facts in PROLOG for The Statements- Gold is valuable. Jane is female. John is the father of Mary. John gives the book to mary.		
	2. To write a LISP Program to solve the water-jug problem using heuristic function.		
	3. To create a compound object using Turbo Prolog.		
	4. To write a Prolog Prog		
3	1. Write a Prolog program to use of conjunctions.		
	2. Write any three programs using conjunction and Disjunctions		
4.	1. Construct a database consisting of facts about a family. make the predicates statements and rules with arguments and satisfy different queries.		
5.	1. Construct a database consisting of facts and consider a rule: a person may steal something if the person is a thief and the person likes the thing. What happens when the question What may john steal? is asked		
6.	1. Write a program in prolog to implement simple arithmetic.		

7.	1. Write a program to implement Factorial of a given number.		
8.	1. Write a program to implement Fibonacci of a given number.		
9.	1. Write Prolog implementation of Breadth First Search.		
	2. Write a program to illustrate the depth first search in Prolog		
	3. Write prolog program to solve the monkey banana problem		
	4. Write the predicate that reverses the elements of the list.		
	5. Write a program to implementation of problem-solving strategies: Forward Chaining & Backward Chaining		
	6. Write a Prolog program to solve Water Jug problem using heuristic function.		
	7. Write a Prolog program to solve N queen Problem.		
	8. Write a Prolog program to implement traveling salesman problem.		

INSTRUCTIONS TO STUDENTS

Before entering the lab the student should carry the following things

(MANDATORY)

1. Identity card issued by the college.
 2. Class notes
 3. Lab observation book
 4. Lab Manual
 5. Lab Record
- Student must sign in and sign out in the register provided when attending the lab session without fail.
 - Come to the laboratory in time. Students, who are late more than 15 min., will not be allowed to attend the lab.
 - Students need to maintain 100% attendance in lab if not a strict action will be taken.
 - All students must follow a Dress Code while in the laboratory.
 - All bags must be left at the indicated place.
 - Refer to the lab staff if you need any help in using the lab.
 - Respect the laboratory and its other users.
 - Work space must be kept clean and tidy after experiment is completed.
 - Read the Manual carefully before coming to the laboratory and be sure about what you are supposed to do.
 - Do the experiments as per the instructions given in the manual.
 - Copy all the programs to observation which are taught in class before attending the lab session.
 - Students are not supposed to use pen drives without permission of lab- in charge.
 - Lab records need to be submitted on or before the date of submission.

LABEXERCISE#1

Objective(s):

To be familiar with syntax and structure of Prolog programming.

Program: Study of PROLOG Programming language and its Functions. Write simple facts for the statements using PROLOG.

Code: *(Use comments where applicable)*

PROGRAM:

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

OUTPUT:

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

LABEXERCISE#2

Objective(s): To be familiar with different Facts and clauses in prolog.

Program: Write the facts in PROLOG for The Statements- Gold is valuable. Jane is female. John is the father of Mary. John gives the book to mary. Evaluate the queries using facts created.

Code: *(Use comments where ever applicable)*

```
valuable(gold).  
female(jane).  
father(john,marry).  
gives( Books,john,marry).
```

SAMPLEPROGRAMS

(Students are to code the following programs in the lab and show the output to instructor)

Instructions

- *Write comment to make your programs readable.*
- *Use descriptive variables in your programs(Name of the variables should show their purposes)*

Programs List

1. To write a LISP Program to solve the water-jug problem using heuristic function.
2. To create a compound objects using Turbo Prolog.
3. To write a Prolog Prog

LABEXERCISE#3

Objective(s):

To understand the use of conjunctions in Prolog

Program: Write a Prolog program to use of conjunctions

Code:*(Use comments wherever applicable)*

% Example of conjunction (AND) happy(john), sunny_day.

% Example of disjunction (OR) likes(john, ice_cream) ; likes(john, pizza).

% Example of combining conjunction and disjunction happy(john) ; (sunny_day, not(busy(john))).

% Example of negation (NOT) rainy_day :- not(sunny_day).

SAMPLEPROGRAMS

(Students are to code the following programs in the lab and show the output to instructor)

Instructions

- *Write comment to make your programs readable.*
- *Use descriptive variables in your programs(Name of the variables should show their purposes)*

Programs List

Write any three programs using conjunction and Disjunctions.

LABEXERCISE#4

Objective(s):

To understand the programming using Facts Predicates and rules.

Program: Construct a database consisting of facts about a family. make the predicates statements and rules with arguments and satisfy different queries.

Code:

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

Queries:

```
mother_of(jane, paul).
mother_of(jane, mary).
```

```
male(paul).
male(joe).
```

```
female(mary).
female(jane).
```

```
father_of(joe, paul).
father_of(joe, mary).
```

```
(10 ms) yes
| ?- father_of(joe,paul).
```

```
true ?
```

```
yes
| ?- father_of(paul,mary).
```

```
no
| ?- father_of(X,mary).
```

```
X = joe
```

```
yes
```

SAMPLEPROGRAMS

(Students are to code the following programs in the lab and how the output to instructor)

Instructions

- Write comment to make your programs readable.
- Use descriptive variables in your programs(Name of the variables should how their purposes)

ProgramsList

```
female(pammi).
female(lizza).
female(patty).
female(anny).
male(jimmy).
male(bobby).
male(tomy).
male(pitter).
parent(pammi,bobby).
parent(tomy,bobby).
parent(tomy,lizza).
parent(bobby,anny).
parent(bobby,patty).
parent(patty,jimmy).
parent(bobby,pitter).
parent(pitter,jimmy).
mother(X,Y):- parent(X,Y),female(X).
father(X,Y):- parent(X,Y),male(X).
haschild(X):- parent(X,_).
sister(X,Y):- parent(Z,X),parent(Z,Y),female(X),X\==Y.
brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.
Input:
parent(X,jimmy).
mother(X,Y).
haschild(X).
sister(X,Y).
```

Output?

LABEXERCISE#5

Objective(s):

To understand matching a rule.

Program: Construct a database consisting of facts and consider a rule: a person may steal something if the person is a thief and the person likes the thing. What happens when the question What may john steal? is asked.

Code:

Matching a Rule A person may steal something if the person is a thief and the person likes the thing.

•Consider the following database: thief(john). likes(joe,food). likes(joe,wine). likes(john,X):-likes (X,wine). kidnap(X, Y):-thief (X), likes ? - kidnap(john, X). X = joe

LABEXERCISE#6

Objective(s):

To understand use of Arithmetic operations in prolog

Program: Write a program in prolog to implement simple arithmetic.

Code:

Arithmetic examples	Prolog Notation
$6 + 2 = 8$	8 is 6+2.
$6 * 2 = 12$	12 is 6*2.
$6 - 2 = 4$	4 is 6-2.
$6 - 8 = -2$	-2 is 6-8.
$6 \div 2 = 3$	3 is 6/2.
$7 \div 2 = 3$	3 is 7/2.
1 is the remainder when 7 is divided by 2 1 is mod(7,2).	

Posing the following queries yields the following responses:

?- 8 is 6+2.

yes

?- 12 is 6*2.

yes

?- -2 is 6-8.

yes

?- 3 is 6/2.

yes

?- 1 is mod(7,2).

yes

More importantly, we can work out the answers to arithmetic questions by using variables. For example:

?- X is 6+2.

X = 8

?- X is 6*2.

X = 12

?- R is mod(7,2).

R = 1

LABEXERCISE#7

Objective(s):

To understand programming with recursive functions.

Program: Write a program to implement Factorial of a given number.

Code:

```
fact(0,1).
fact(N,F):-
(
    % The below is for +ve factorial.
    N>0 ->
    (
        N1 is N-1,
        fact(N1,F1),
        F is N*F1
    )
    ;

    % The below is for -ve factorial.
    N<0 ->
    (
        N1 is N+1,
        fact(N1,F1),
        F is N*F1
    )
).

```

LABEXERCISE#8

Objective(s):

To understand the concept of Fibonacci Series.

Program1: Write a program to implement Fibonacci of a given number.

Code:

```
fib(0,  
1) :-  
!.
```

```
fib(1, 1) :- !.
```

```
fib(N, Result) :- N1 is N - 1, N2 is N - 2, fib(N1, Result1), fib(N2, Result2), Result is  
Result1 + Result2.
```

ASSIGNMENT #2

LAB EXERCISE

Objective(s):

To understand the searching techniques in prolog

Program1: Write Prolog implementation of Breadth First Search.

Code:

```
% Define edges of the graph: edge(Node1, Node2).
```

```
edge(a, b).
```

```
edge(a, c).
```

```
edge(b, d).
```

```
edge(c, d).
```

```
edge(c, e).
```

```
edge(d, f).
```

```
edge(e, f).
```

```
% Bidirectional edges (if the graph is undirected)
```

```
connected(X, Y) :- edge(X, Y).
```

```
connected(X, Y) :- edge(Y, X).
```

```
% BFS driver: find a path from Start to Goal
```

```
bfs(Start, Goal, Path) :-
```

```
    bfs_helper([[Start]], Goal, RevPath),
```

```
    reverse(RevPath, Path).
```

```
% Helper for BFS
```

```
bfs_helper([[Goal|RestPath]|_], Goal, [Goal|RestPath]).
```

Program2: Write a program to illustrate the depth first search in Prolog

Code:

```
% Define edges of the graph: edge(Node1, Node2).
edge(a, b).
edge(a, c).
edge(b, d).
edge(c, d).
edge(c, e).
edge(d, f).
edge(e, f).
```

```
% Bidirectional edges (if the graph is undirected)
connected(X, Y) :- edge(X, Y).
connected(X, Y) :- edge(Y, X).
```

```
% DFS driver: find a path from Start to Goal
dfs(Start, Goal, Path) :-
    dfs_helper(Start, Goal, [Start], RevPath),
    reverse(RevPath, Path).
```

```
% Helper for DFS
dfs_helper(Goal, Goal, Path, Path).
dfs_helper(Current, Goal, Visited, Path) :-
    connected(Current, Next),
    \+ member(Next, Visited),
    dfs_helper(Next, Goal, [Next|Visited], Path).
```



Example Query

To find a path from a to f, you would query:

```
prolog
Copy
Edit
?- dfs(a, f, Path).
Expected output:
```

```
prolog
Copy
Edit
Path = [a, b, d, f] ;
Path = [a, c, d, f] ;
Path = [a, c, e, f] ;
false.
```

Program 3. Write prolog program to solve the monkey banana problem

Code:

```
move(state(middle, onbox, middle, hasnot),
      grasp,
      state(middle, onbox, middle, has)).

move(state(P, onfloor, P, H),
      climb,
      state(P, onbox, P, H)).

move(state(P1, onfloor, P1, H),
      push(P1, P2),
      state(P2, onfloor, P2, H)).

move(state(P1, onfloor, B, H),
      walk(P1, P2),
      state(P2, onfloor, B, H)).

% Define goal state
canget(state(_, _, _, has)).

% Define recursive rule to reach goal state
canget(State1) :-
    move(State1, _, State2),
    canget(State2).
```

Program 4. Write the predicate that reverses the elements of the list.

Code:

Method 1

naïve Reverse

```
naive_reverse([], []).
naive_reverse([Head|Tail], Reversed) :-
    naive_reverse(Tail, RevTail),
    append(RevTail, [Head], Reversed).
```

Method 2. Using Accumulator

```
reverse(List, Reversed) :-
    reverse_helper(List, [], Reversed).

reverse_helper([], Accumulator, Accumulator).
reverse_helper([Head|Tail], Accumulator, Reversed) :-
    reverse_helper(Tail, [Head|Accumulator], Reversed).
```

Program 5. Write a program to implementation of problem-solving strategies:
Forward Chaining & Backward Chaining

Code:

```
% Check if a fact is already known
known(Fact, KnownFacts) :-
    member(Fact, KnownFacts).

% Apply rules to derive new facts
forward_chain(KnownFacts, NewFacts) :-
    findall(NewFact,
        (rule(NewFact, Conditions),
         \+ known(NewFact, KnownFacts),
         all_conditions_met(Conditions, KnownFacts)),
        DerivedFacts),
    ( DerivedFacts \= []
    -> append(KnownFacts, DerivedFacts, UpdatedFacts),
        forward_chain(UpdatedFacts, NewFacts)
    ; NewFacts = KnownFacts
    ).

% Check if all conditions of a rule are met
all_conditions_met([], _).
all_conditions_met([Condition|Rest], KnownFacts) :-
    known(Condition, KnownFacts),
    all_conditions_met(Rest, KnownFacts).
```

Usage:

```
prolog
Copy
Edit
?- forward_chain([cloudy, humid], AllFacts).
AllFacts = [cloudy, humid, rain, wet_ground].
```

Backword Chaining

```
% Facts
cloudy.
windy.

% Rules
rainy :- cloudy.
stormy :- rainy, windy.
good_day_for_picnic :- sunny.

% Additional Rule
sunny :- \+ cloudy.
```

Program 6. Write a Prolog program to solve Water Jug problem using heuristic function.
Code:

```
% Define the goal state
goal(jug(2, _)).

% Define possible actions
action(fill_x, jug(X, Y), jug(4, Y)) :- X < 4.
action(fill_y, jug(X, Y), jug(X, 3)) :- Y < 3.
action(empty_x, jug(X, Y), jug(0, Y)) :- X > 0.
action(empty_y, jug(X, Y), jug(X, 0)) :- Y > 0.
action(pour_x_to_y, jug(X, Y), jug(NewX, NewY)) :-
    X > 0, Y < 3,
    Transfer is min(X, 3 - Y),
    NewX is X - Transfer,
    NewY is Y + Transfer.
action(pour_y_to_x, jug(X, Y), jug(NewX, NewY)) :-
    Y > 0, X < 4,
    Transfer is min(Y, 4 - X),
    NewX is X + Transfer,
    NewY is Y - Transfer.

% Heuristic function
heuristic(jug(X, _), H) :-
    H is abs(2 - X).

% A* search algorithm
a_star([[State, Path, Cost] | _], Path) :-
    goal(State).
a_star([[State, Path, Cost] | Rest], SolutionPath) :-
    findall([NextState, [Action | Path], NewCost],
        (action(Action, State, NextState),
         \+ member([NextState, _, _], Rest),
         \+ member(NextState, Path),
         NewCost is Cost + 1),
        NextMoves),
    append(Rest, NextMoves, NewQueue),
    sort_queue(NewQueue, SortedQueue),
    a_star(SortedQueue, SolutionPath).

% Sort the queue based on the estimated total cost (Cost + Heuristic)
sort_queue(Queue, SortedQueue) :-
    map_list_to_pairs(estimate_total_cost, Queue, Pairs),
```

```

keysort(Pairs, SortedPairs),
pairs_values(SortedPairs, SortedQueue).

% Estimate total cost for A* (Cost so far + Heuristic)
estimate_total_cost([State, _, Cost], TotalCost) :-
    heuristic(State, H),
    TotalCost is Cost + H.

% Solve the problem
solve(Path) :-
    a_star([[jug(0, 0), [], 0]], RevPath),
    reverse(RevPath, Path).

```

Program 7. Write a Prolog program to solve N queen Problem.
Code:

```

n_queens(N, Queens) :-
    length(Queens, N),
    Queens ins 1..N,
    all_distinct(Queens),
    safe_queens(Queens),
    labeling([ff], Queens).

safe_queens([]).
safe_queens([Q|Qs]) :-
    safe_queens(Qs, Q, 1),
    safe_queens(Qs).

safe_queens([], _, _).
safe_queens([Q|Qs], Q0, D0) :-
    Q0 #\= Q,
    abs(Q0 - Q) #\= D0,
    D1 #= D0 + 1,
    safe_queens(Qs, Q0, D1).

```

Program 8. Write a Prolog program to implement traveling salesman problem.

Code:

```
% Define distances between cities
distance(city1, city2, 10).
distance(city2, city3, 15).
distance(city3, city4, 20).
distance(city4, city1, 25).
% (Add all necessary distance facts)

% Generate all possible routes
route([City], [City], 0).
route([City1, City2 | Rest], [City1, City2 | Rest], Distance) :-
    distance(City1, City2, D),
    route([City2 | Rest], [City2 | Rest], D1),
    Distance is D + D1.

% Find the shortest route
shortest_route(Route, Distance) :-
    findall([R, D], route(R, [], D), Routes),
    sort(2, @=<, Routes, [[Route, Distance] | _]).

% Example query
% ?- shortest_route(Route, Distance).
% Route = [city1, city2, city3, city4],
% Distance = 60.
```