ATMA RAM SANATAN
DHARMA COLLEGE

Artificial Intelligence
Practical File

Name - Shreyan Mohapatra

Roll No - 24/48036

Course - B.Sc. (Hons) Computer Science

INDEX

S. No.	Practical Number	Date
1	Practical 1	10 August, 2025
2	Practical 2	14 August, 2025
3	Practical 3	21 August, 2025
4	Practical 4	21 August, 2025
5	Practical 5	28 August, 2025
6	Practical 6	28 August, 2025
7	Practical 7	11 September, 2025
8	Practical 8	11 September, 2025
9	Practical 9	9 October, 2025
10	Practical 10	9 October, 2025
11	Practical 11	30 October, 2025
12	Practical 12	30 October, 2025
13	Practical 13	6 November, 2025
14	Practical 14	6 November, 2025
15	Practical 15	6 November, 2025

1. Write a PROLOG program to implement the family tree and demonstrate the family

relationship.

Program For Dashrath's Family Tree :

dashrath.pl [modified]

```
/* ---------- Dashrath's Family Tree ---------- */

% Gender facts
male(dashrath).
male(rama).
male(bharat).
male(lakshman).
male(shatrughna).
male(luv).
male(kush).

female(kaushalya).
female(kaikeyi).
female(sumitra).
female(sita).
female(urmila).

% Parent relationships
parent(dashrath, rama).
parent(dashrath, bharat).
parent(dashrath, lakshman).
parent(dashrath, shatrughna).

parent(kaushalya, rama).
parent(kaikeyi, bharat).
parent(sumitra, lakshman).
parent(sumitra, shatrughna).

parent(rama, luv).
parent(rama, kush).
parent(sita, luv).
parent(sita, kush).
```

```
/* ---------- Relationship Rules ---------- */

% Father and Mother
father(F, C) :- male(F), parent(F, C).
mother(M, C) :- female(M), parent(M, C).

% Siblings (share at least one parent)
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.

% Grandparent
grandparent(GP, GC) :- parent(GP, P), parent(P, GC).
grandfather(GF, GC) :- male(GF), grandparent(GF, GC).
grandmother(GM, GC) :- female(GM), grandparent(GM, GC).

% Husband/Wife
husband(H, W) :- married(H, W), male(H).
wife(W, H) :- married(H, W), female(W).
```

IMPLEMENTATION IN PROLOG :

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [dashrath].
true.

?- father(dashrath, rama).
true .

?- grandfather(dashrath, kush).
true .

?- sibling(rama, sita).
false.

?-
```

2. Write a Prolog program to implement conc(L1, L2, L3) where L2 is the first to be appended with L1to get resulted list L3.

CODE:

concatenate.pl

```
1 % Base case:
2 conc([], L2, L2).
3
4 % Recursive case:
5 conc([Head|L1_Tail], L2, [Head|L3_Tail]) :-
6     conc(L1_Tail, L2, L3_Tail).
7
```

Output:

```
?- set_prolog_flag(prompt_alternatives_on, groundness).

?- consult('C:/Users/arpit/OneDrive/Desktop/concatenate.pl').

?- conc([a,b], [c,d], L3).
L3 = [a, b, c, d] ;
false.

?- conc(L1, L2, [1,2,3,4,5]).
L1 = [],
L2 = [1, 2, 3, 4, 5] ;
L1 = [1],
L2 = [2, 3, 4, 5] ;
L1 = [1, 2],
L2 = [3, 4, 5] ;
L1 = [1, 2, 3],
L2 = [4, 5] ;
L1 = [1, 2, 3, 4],
L2 = [5] ;
L1 = [1, 2, 3, 4, 5],
L2 = [] ;
false.

?- conc([7,8,0], L2, [7,8,0,e,f,g,h]).
L2 = [e, f, g, h] ;
false.

?-
```

3. Write a prolog program to implement reverse(L,R) where list L is original and List R is reversed list.

reverse.pl [modified]

```
% reverse(L, R) :- R is the reverse of list L.

% Base case: reversing an empty list gives an empty list.
reverse([], []).

% Recursive case:
% Reverse the tail (T) into RT,
% then append the head (H) at the end.
reverse([H|T], R) :-
    reverse(T, RT),
    append(RT, [H], R).

% append(X, Y, Z) :- appends lists X and Y into Z.
append([], L, L).
append([H|T], L, [H|R]) :-
    append(T, L, R).
```

IMPLEMENTATION:

SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [reverse].
true.

?- reverse([1,2,3,4,5],R).
R = [5, 4, 3, 2, 1].

?- reverse([a,b,c,d],R).
R = [d, c, b, a].

?-
```

4. Write a prolog program to calculate the sum of two numbers.

sum.pl

```
% sum(X, Y, Z) :- Z is the sum of X and Y.

sum(X, Y, Z) :-
    Z is X + Y.
```

IMPLEMENTATION :

```
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [sum].
true.

?- sum(2,3,R).
R = 5.

?- sum(5,7,Z).
Z = 12.

?- sum(3,8,11).
true.

?- sum(3,5,7).
false.

?-
```

OUTPUT:

5. Write a PROLOG program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.

Aim:

To write a Prolog program that finds the maximum of two given numbers.

Theory:

In Prolog, rules and facts are used instead of conventional control structures. To find the maximum of two numbers, we can compare them using relational operators (>=, <) and assign the larger one as the result. This demonstrates the use of condition checking and decision-making in logic programming.

Program Used: SWI Prolog

CODE:

File Edit Browse Compile Prolog Pce Help

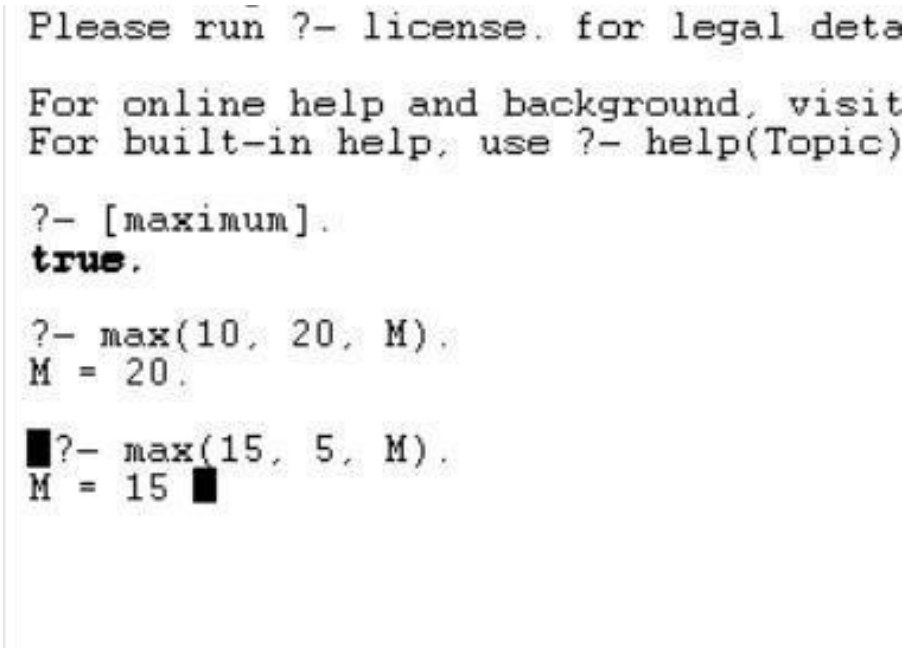
maximum.pl [modified]

```
% Prolog program to find maximum of two numbers

max(X, Y, M) :-
    X >= Y,
    M is X.

max(X, Y, M) :-
    X < Y,
    M is Y.
```

PRACTICAL-5



Lesson Learned:

Learned how to implement decision-making in Prolog.

Understood the use of comparison operators in Prolog.

Practiced defining rules that handle multiple cases.

OUTPUT :

Aim:

To write a Prolog program that computes the factorial of a number using recursion.

Theory:

Factorial of a number N is defined as:

0! = 1 (base case)

N! = N × (N-1)! for N > 0

Recursion is a fundamental concept in both mathematics and logic programming. In Prolog, recursion is expressed by defining a base case and a recursive rule that reduces the problem size step by step until the base case is reached.

Program Used: SWI Prolog

CODE:

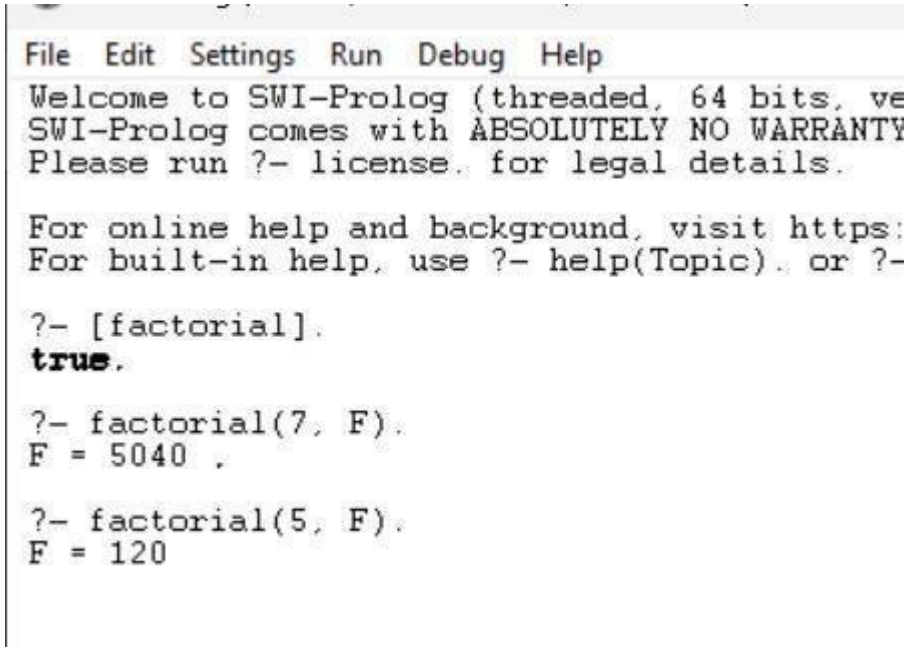
File Edit Browse Compile Prolog Pce Help

factorial.pl [modified]

```
% Base case
factorial(0, 1).

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

PRACTICAL-6



Lesson Learned:

Learned how to implement decision-making in Prolog.

Understood the use of comparison operators in Prolog.

Practiced defining rules that handle multiple cases.

OUTPUT:

Aim:

Write a Prolog program to compute the Nth term of the Fibonacci series.

Theory:

The Fibonacci sequence is defined as:

F(0) = 0

F(1) = 1

F(N) = F(N-1) + F(N-2) for N > 1

Recursion is used to compute the Nth Fibonacci number. The program follows the same principle as mathematical recursion: a base case for N = 0 and N = 1, and a recursive case for N > 1.

Program Used: SWI Prolog

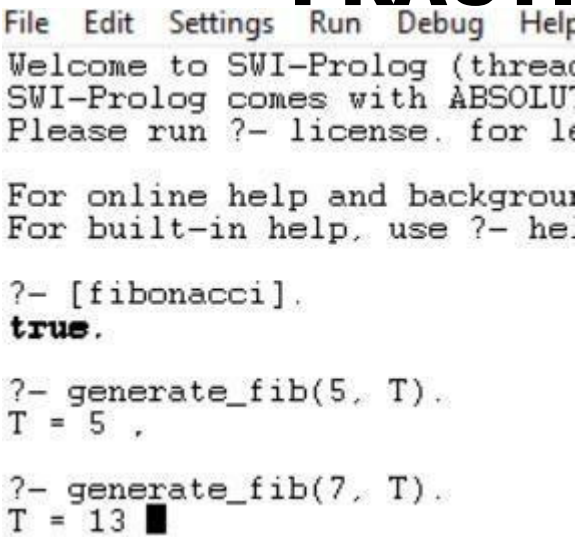
CODE:

fibonacci.pl [modified]

```
% Base cases
generate_fib(0, 0).
generate_fib(1, 1).

% Recursive case
generate_fib(N, T) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    generate_fib(N1, T1),
    generate_fib(N2, T2),
    T is T1 + T2.
```

PRACTICAL-7



Lesson Learned:

Understood how to use recursion in Prolog for sequence generation.

Learned the importance of defining correct base cases and recursive cases.

Practiced solving problems by breaking them into smaller subproblems.

OUTPUT:

Aim:

Write a Prolog program to compute Num raised to the power Pow.

Theory:

The power of a number is calculated as:

Num^0 = 1 (base case)

Num^Pow = Num * Num^(Pow-1) for Pow > 0

Recursion is used to compute power. The base case handles the zero exponent. The recursive rule multiplies Num by the result of power(Num, Pow-1, Ans).

Program Used: SWI Prolog

CODE:

power

```
% Base case
power(_, 0, 1).

% Recursive case
power(Num, Pow, Ans) :-
    Pow > 0,
    P1 is Pow - 1,
    power(Num, P1, Ans1),
    Ans is Num * Ans1.
```

PRACTICAL-8

```
File  Edit  Settings  Run  Debug  Help
SWI-Prolog comes with ABSOLUTELY
Please run ?- license. for legal

For online help and background,
For built-in help, use ?- help(T

?- [power].
true.

?- power(2, 3, Ans).
Ans = 8 ,

?- power(5, 0, Ans).
Ans = 1
```

Lesson Learned:

Learned to implement exponentiation using recursion in Prolog.

Practiced using arithmetic expressions like is and decrementing the power.

Understood the structure of recursive logic programs and importance of the base case.

OUTPUT:

Aim:

To write a Prolog program that multiplies two numbers N1 and N2, and stores the result in R.

Theory:

In Prolog, arithmetic operations are performed using the is operator. Multiplication of two numbers can be done directly using R is N1 * N2. This program demonstrates the use of arithmetic expressions and simple fact–rule definition in logic programming.

Program Used: SWI Prolog

CODE:

multi

```
% Program to multiply two numbers

multi(N1, N2, R) :-
    R is N1 * N2.
```

PRACTICAL-9

```
?- [multi].
true.

?- multi(5,3,R).
R = 15.

?-
```

Lesson Learned:

Learned how to perform arithmetic operations in Prolog.

Understood the use of the is operator for evaluation.

Practiced creating simple relation-based programs in Prolog.

OUTPUT:

Aim:

To write a Prolog program that checks whether an element X is a member of a list L or not.

Theory:

The member relation checks if an element exists in a list.
This can be implemented recursively:

Base case: If the head of the list equals X, then X is a member.

Recursive case: If not, check whether X is a member of the tail of the list.

This demonstrates the concept of **pattern matching** and **recursion** in Prolog lists.

Program Used: SWI Prolog

CODE:

member

```
% Program to check whether X is a member of list L

memb(X, [X|_]).            % Base case: X is the head of the list
memb(X, [_|Tail]) :-       % Recursive case: check in the tail
    memb(X, Tail).
```

PRACTICAL-10

```
?- [member].
true.

?- memb(3, [1, 2, 3, 4]).
true .

?- memb(7, [2, 4, 6, 8]).
false.

?-
```

Lesson Learned:

Learned how to represent and traverse lists in Prolog.

Understood recursive search through list elements.

Practiced using pattern matching ([Head|Tail]) in logic programming.

Lesson Learned:

Aim:

To write a Prolog program to find the sum of all elements in a given list.

Theory:

In Prolog, recursion is used to traverse a list. The sum of a list can be calculated by:

Adding the head element to the sum of the tail list.

Using a base case for an empty list (sum = 0).

Program Used: SWI Prolog

CODE:

sumlist.pl [modified]

```
% Program to find sum of elements in a list

% Base case

sumlist([], 0).
sumlist([H|T], S) :-
    sumlist(T, S1),
    S is H + S1.
```

OUTPUT:

```
?- [sumlist].
true.

?- sumlist([1, 2, 3, 4, 5], S).
S = 15.
```

PRACTICAL-11

· Learned recursive list processing.

· Understood how to accumulate results using is.

· Practiced base case and recursive case logic.

Lesson Learned:

Aim:

To write Prolog predicates evenlength(List) and oddlength(List) which are true if the list has even or odd length respectively.

Theory:

List length can be determined recursively:

An empty list has even length.

If a list is even, adding one element makes it odd, and vice versa.

Program Used: SWI Prolog

CODE:

length

```
% Program to check even or odd length of list

evenlength([]).                              % Empty list is even
evenlength([_,_|T]) :- evenlength(T).        % Remove two elements and check

oddlength([_]).                              % Single element list is odd
oddlength([_,_|T]) :- oddlength(T).          % Remove two elements and check
```

OUTPUT:

```
?- [length].
true.

?- evenlength([a, b, c, d]).
true.

?- oddlength([1, 2, 3]).
true
```

PRACTICAL-12

- Understood recursive pattern matching on lists.
- Learned alternating even–odd behavior in list processing.
- Practiced logic without arithmetic.

Lesson Learned:

Aim:

To write a Prolog program to find the maximum element M in a given list L.

Theory:

Finding the maximum in a list uses recursion:

The head is compared with the maximum of the tail using the comparison operators > and =<.

The greater value is returned upwards in recursion.

Program Used: SWI Prolog

CODE:

maxlist [modified]

```
% Program to find maximum element in a list

maxlist([X], X).                                    % Base case
maxlist([H|T], M) :-
    maxlist(T, M1),
    (H > M1 -> M = H ; M = M1).
```

OUTPUT:

```
?- [maxlist].
true.

?- maxlist([3, 9, 5, 2, 7], M).
M = 9
```

PRACTICAL-13

- Learned recursive comparison of list elements.
- Practiced use of conditional (-> ;) in Prolog.
- Understood how to carry results up recursion.

Lesson Learned:

Aim:

To write a Prolog program that inserts an element I into the Nth position of a list L to produce a list R.

Theory:

Insertion in a list can be achieved by:

Recursively moving to the Nth position.

Once position 1 is reached, insert the element as the head.

Program Used: SWI Prolog

CODE:

insert

```
% Program to insert element at Nth position

insert(I, 1, L, [I|L]).                    % Insert at first position
insert(I, N, [H|T], [H|R]) :-
    N1 is N - 1,
    insert(I, N1, T, R).
```

OUTPUT:

```
?- [insert].
true.

?- insert(10, 3, [1, 2, 3, 4], R).
R = [1, 2, 10, 3, 4]
```

PRACTICAL-14

- Understood list manipulation using recursion.
- Learned use of arithmetic decrement for position tracking.
- Practiced structural list construction in Prolog.

Lesson Learned:

Aim:

To write a Prolog program that deletes the element at the Nth position from a list L, producing list R.

Theory:

Deletion follows similar logic to insertion:

Traverse recursively to the desired position.

Skip the element at that position to form the new list.

Program Used: SWI Prolog

CODE:

delete

```
% Program to delete element from Nth position

delete(1, [_|T], T).                    % Delete first element
delete(N, [H|T], [H|R]) :-
    N1 is N - 1,
    delete(N1, T, R).
```

OUTPUT:

```
?- [delete].
true.

?- delete(3, [10, 20, 30, 40], R).
R = [10, 20, 40]
```

PRACTICAL-15

Lesson Learned:

· Learned recursive deletion using list decomposition.

· Understood positional tracking via arithmetic operations.

· Practiced list reconstruction techniques in logic programming.