Artificial Intelligence

Practical file

University of Delhi
2025-26

Name : Anjali

Roll No : 24570008

Semester : 3rd

Submitted To : Bhavya Ahuja

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

m.pl

File Edit Browse Compile Prolog Pce Help

m.pl

```prolog
% --- Facts ---
male(john).
female(mary).
male(peter).
female(susan).
male(kevin).
female(anna).

parent(john, peter).
parent(mary, peter).
parent(john, susan).
parent(mary, susan).
parent(peter, kevin).
parent(peter, anna).

% --- Rules ---
father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).
child(X, Y) :- parent(Y, X).
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.
brother(X, Y) :- sibling(X, Y), male(X).
sister(X, Y) :- sibling(X, Y), female(X).
grandparent(X, Z) :- parent(X, Y), parent(Y, Z).
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

SWI-Prolog console

File Settings Tools Help

```
?- mother(mary, susan).
true.

?- sibling(peter, susan).
true ;
true.

?- grandparent(john, kevin).
true ;
false.

?- ancestor(john, anna).
true .

?-
```

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

u.pl

File Edit Browse Compile Prolog Pce Help

u.pl

```prolog
% Base case: if first list is empty, result is the second list
conc([], L, L).

% Recursive case: take head (H) of first list and append recursively
conc([H|T], L2, [H|L3]) :-
    conc(T, L2, L3).
```

```
?- conc([1,2,3], [4,5,6], L3).
L3 = [1, 2, 3, 4, 5, 6].

?- conc([], [a,b], L3).
L3 = [a, b].

?- conc([x,y], [], L3).
L3 = [x, y].

?-
```

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

u.pl

File Edit Browse Compile Prolog Pce Help

u.pl

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

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

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

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

?- reverse([], R).
R = [].
```

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

```
u.pl
File  Edit  Browse  Compile  Prolog  Pce  Help
u.pl
% sum(X, Y, Z) means Z is the sum of X and Y
sum(X, Y, Z) :-
    Z is X + Y.
```

```
?- sum(5, 3, Z).
Z = 8.

?- sum(10, 20, R).
R = 30.

?- sum(-2, 7, S).
S = 5.
```

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

```
u.pl
File  Edit  Browse  Compile  Prolog  Pce  Help
u.pl
% If X is greater than or equal to Y, then M = X
max(X, Y, X) :-
    X >= Y.

% Otherwise, M = Y
max(X, Y, Y) :-
    X < Y.
```

```
?- max(5, 8, M).
M = 8.

?- max(12, 4, M).
M = 12 .
```

6. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.

u.pl

File Edit Browse Compile Prolog Pce Help

u.pl

```prolog
% Base case: factorial of 0 is 1
factorial(0, 1).

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

SWI-Prolog console

File Settings Tools Help

```
F = 120 .

?- factorial(0, F).
F = 1 .

?- factorial(3, F).
F = 6
```

7. Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the Fibonacci series.

u.pl

```prolog
% 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.
```

```
?- generate_fib(7, T).
T = 13 .

?- generate_fib(0, T).
T = 0 .

?-
```

8. Write a PROLOG program to implement power (Num, Pow, Ans) : where Num is raised to the power Pow to get Ans.

k.pl

```
% Base case: any number raised to power 0 is 1
power(_, 0, 1).

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

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

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

?- power(3, 4, Ans).
Ans = 81 .

?-
```

9. PROLOG program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.

k.pl

```
% Rule: R is the product of N1 and N2
multi(N1, N2, R) :-
    R is N1 * N2.
```

```
?- multi(5,10,R).
R = 50.

?- multi(0,1,R).
R = 0.

?- multi(8,9,R).
R = 72.

?-
```

10. Write a PROLOG program to implement memb(X, L): to check whether X is a member of L or not.

k.pl

```
% Base case: X is the head of the list
memb(X, [X|_]).

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

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

?- memb(a, [b,c,d]).
false.

?- memb(x, [x,y,z]).
true
```

11. Write a PROLOG program to implement sumlist(L, S) so that S is the sum of a given list L.

```
File  Edit  Browse  Compile  Prolog  Pce  Help
k.pl
% Base case: sum of an empty list is 0
sumlist([], 0).

% Recursive case: add head to the sum of tail
sumlist([H|T], S) :-
    sumlist(T, Rest),
    S is H + Rest.
```

```
?- sumlist([1,2,3,4], S).
S = 10.

?- sumlist([5,10,15], S).
S = 30.
```

12. Write a PROLOG program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively

```
k.pl [modified]
% Base cases
evenlength([]).          % Empty list has even length
oddlength([_]).

% Recursive cases
evenlength([_,_|T]) :-    % Remove two elements each time
    evenlength(T).

oddlength([_,_|T]) :-     % Remove two elements and check again
    oddlength(T).
```

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

?- evenlength([x,y,z]).
false.

?- oddlength([x,y,z,w]).
false.
```

13. Write a PROLOG program to implement maxlist(L, M) so that M is the maximum number in the list.

k.pl [modified]

```
% Base case: if only one element, it's the maximum
maxlist([X], X).

% Recursive case 1: if head >= max of tail
maxlist([H|T], H) :-
    maxlist(T, M),
    H >= M.

% Recursive case 2: if max of tail > head
maxlist([H|T], M) :-
    maxlist(T, M),
    H < M.
```

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

?- maxlist([10,4,6], M).
M = 10 .
```

14. Write a PROLOG program to implement insert(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.

```
% Base case: Insert at position 1 (beginning of the list)
insert(I, 1, L, [I|L]).

% Recursive case: Move forward in the list until position 1 is reached
insert(I, N, [H|T], [H|R]) :-
    N > 1,
    N1 is N - 1,
    insert(I, N1, T, R).
```

```
?- insert(x, 1, [a,b,c], R).
R = [x, a, b, c] .

?- insert(x, 3, [a,b,c,d], R).
R = [a, b, x, c, d] .

?- insert(10, 5, [1,2,3,4,6], R).
R = [1, 2, 3, 4, 10, 6] .

?-
```

15. Write a PROLOG program to implement delete(N, L, R) that removes the element on Nth position from a list L to generate a list R.

```
% delete(N, L, R)
% Deletes the element at the Nth position in list L and returns the result in R.

delete(1, [_|T], T).        % Base case: If N=1, remove the head of the list.
delete(N, [H|T], [H|R]) :-
    N > 1,                  % Ensure N is greater than 1
    N1 is N - 1,            % Decrement position counter
    delete(N1, T, R).       % Recursive call
```

```
?- delete(2, [a, b, c, d], R).
R = [a, c, d] .

?- delete(1, [x, y, z], R).
R = [y, z] .

?- delete(4, [10, 20, 30, 40, 50], R).
R = [10, 20, 30, 50] .
```