PROLOG Practical File

ARTIFICIAL INTELLIGENCE

Submitted to:- Mr. Ankit Rajpal

Surbhi Mittal
14HCS4101
B.Sc (H) Computer Science

1. **Write a PROLOG program to implement a family-tree.**

```
is_father(X) :-   male(X),
                  father(X,_).

is_mother(X) :-   female(X),
                  mother(X,_).

is_son(X) :-      male(X),
                  father(_,X),!.

is_son(X) :-      male(X),
                  mother(_,X).

sibling(X,Y) :-   father(X1,X),
                  father(X1,Y),
                  mother(X2,X),
                  mother(X2,Y).

brother_of(X,Y):- male(X),
                  sibling(X,Y).

sister_of(X,Y):-  female(X),
                  sibling(X,Y).

grandpa_of(X,Y):- father(X1,X),
                  father(X1,Y).

grandson(X,Y) :-  male(X),
                  father(X1,X),
                  father(Y,X1),!.

grandson(X,Y) :-  male(X),
                  mother(X1,X),
                  mother(Y,X1).

first_cousin(X,Y):-father(X1,X),
                   father(X2,Y),
                   brother_of(X1,X2),X\=Y,!.

first_cousin(X,Y):-mother(X1,X),
                   mother(X2,Y),
                   sister_of(X1,X2),X\=Y,!.

first_cousin(X,Y):-father(X1,X),
                   mother(X2,Y),
                   sibling(X1,X2),X\=Y,!.

first_cousin(X,Y):-mother(X1,X),
                   father(X2,Y),
                   sibling(X1,X2),X\=Y.

descendent(X,Y):- father(Y,X),!.
descendent(X,Y):- father(Y1,X),
                  descendent(Y1,Y).
```

```
father(edward,albert).
father(albert,john).
father(albert,bob).
father(albert,mary).
father(albert,jasmine).
father(albert,mahsa).
father(bob,harry).
father(john,arun).
mother(asha,victoria).
mother(victoria,john).
mother(victoria,bob).
mother(victoria,jasmine).
mother(victoria,mary).
mother(victoria,mahsa).
mother(jasmine,pooja).
mother(mary,simi).
female(asha).
female(victoria).
female(mary).
female(mahsa).
female(jasmine).
female(pooja).
female(simi).
male(edward).
male(albert).
male(john).
male(bob).
male(arun).
male(harry).
```

```
1 ?- is_father(edward).
true.

2 ?- is_father(arun).
false.

3 ?- is_mother(victoria).
true .

4 ?- is_mother(simi).
false.

5 ?- descendent(harry,albert).
true.

6 ?-
```

2. **Using PROLOG, write a series of facts and rules that asserts the facts that Bob and Mary speak Russian and John and Mary speak English. It also defines the relation "understands" between two persons, which is true exactly when they both speak the same language. Your program should answer the following queries:**
    a. **?- speaks(X, Russian).**
    b. **?- understands(John, Bob).**

c. **?- understands(X, Bob).**

d. **?- understands(P1, P2).**

```
speaks(bob,russian).
speaks(mary,russian).
speaks(mary,english).
speaks(john,english).

understands(P1,P2):- speaks(P1,X),speaks(P2,X),P1\=P2.
```

```
1 ?- speaks(X,russian).
X = bob ;
X = mary.

2 ?- understands(john,bob).
false.

3 ?- understands(X,bob).
X = mary ;
false.

4 ?- understands(P1,P2).
P1 = bob,
P2 = mary ;
P1 = mary,
P2 = bob ;
P1 = mary,
P2 = john ;
P1 = john,
P2 = mary ;
false.

5 ?-
```

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

```
go:-
    nl,write('Enter first number : '),read(A),
    write('Enter second number: '),read(B),
    sum(A,B,Sum),
    write('Sum : '),write(Sum),nl.

sum(A,B,Sum):- Sum is A+B.
```

```
1 ?- go.

Enter first number : 5.
Enter second number: |: 13.
Sum : 18
true.

2 ?-
```

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

```
go:-
    nl,write('Enter first number : '),read(A),
    write('Enter second number: '),read(B),
    max(A,B,Res),
    write('Maximum : '),write(Res),nl.

max(X,Y,M):- X>Y,X=M,!.
max(_,Y,Y).
```

```
1 ?- go.

Enter first number : 8.
Enter second number: |: 3.
Maximum : 8
true.

2 ?- go.

Enter first number : 2.
Enter second number: |: 12.
Maximum : 12
true.

3 ?-
```

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

```
go:-
 nl,write('Enter a number: '),
 read(N),N>0,
 factorial(N,Res),
 write('Factorial of '),write(N),write(': '),
 write(Res),nl,!.

go:- write('Error!'),nl.

factorial(0,1):-!.
factorial(N,Fact):- N1 is N-1,
                factorial(N1,F1),
                Fact is N*F1.
```

```
1 ?- go.

Enter a number: 5.
Factorial of 5: 120
true.

2 ?- go.

Enter a number: 6.
Factorial of 6: 720
true.

3 ?-
```

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

```
go:-
 nl,write('Enter number : '),
 read(N),nl,
 forLoop(1,N),nl.


forLoop(I,N) :- I=<N,
            fibo(I,Res),
            write(Res),
            write(' '),
            I1 is I+1,
            forLoop(I1,N).

fibo(1,0) :- !.
fibo(2,1) :- !.
fibo(N,T) :- N1 is N-1, N2 is N-2, fibo(N1,R1),fibo(N2,R2), T is R1+R2.
```

```
Enter number : 7.

0 1 1 2 3 5 8
false.

2 ?- go.

Enter number : 3.

0 1 1
false.

3 ?-
```

7. Write a PROLOG program to implement GCD of two numbers.

```
go:-
 nl,write('Enter first number  : '),
 read(A),
 write('Enter second number : '),
 read(B),
 write('GCD                 : '),
 gcd(A,B,Res),write(Res).


gcd(X,0,X) :- !.
gcd(X,Y,Res) :- R1 is X mod Y, gcd(Y,R1,Res).

/*
 *  int gcd(int a, int b)
 *  {
 *    if(b==0)
 *       return a;
 *    else
 *       return gcd(b,a%b);
 */
```

```
1 ?- go.

Enter first number  : 9.
Enter second number : |: 3.
GCD                 : 3
true.

2 ?- go.

Enter first number  : 13.
Enter second number : |: 7.
GCD                 : 1
true.

3 ?-
```

8. Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] where edge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a PROLOG program to check whether there is a route from one node to another node.

```
go:-
    nl,write('Enter starting node : '),
    read(A),
    write('Enter finishing node: '),
    read(B),
    route(A,B),nl,
    write('Path exists.'),nl,!.
```

```
go:-
    nl,write('No path exists.'),nl.


route(A,B) :- edge(A,B),!.
route(A,B) :- edge(A,X), route(X,B).

edge(p,q).
edge(q,r).
edge(q,s).
edge(s,t).
```

```
1 ?- go.

Enter starting node : p.
Enter finishing node: |: t.

Path exists.
true.

2 ?- go.

Enter starting node : r.
Enter finishing node: |: t.

No path exists.
true.

3 ?-
```

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

```
go:-
 nl,write('Enter number  : '),
 read(A),
 write('Enter exponent: '),
 read(B),
 power(A,B,X),nl,
 write(A),write('^'),write(B),write('  : '),
 write(X).

power(_,0,1):-!.
power(X,1,X):-!.
power(X,Y,Res):- N is Y-1, power(X,N,R), Res is R*X.
```

```
1 ?- go.

Enter number   : 4.
Enter exponent: |: 2.

4^2   : 16
true.

2 ?- go.

Enter number   : 3.
Enter exponent: |: 0.

3^0   : 1
true.

3 ?-
```

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

```
go:-
 nl,write('Enter first number : '),
 read(A),
 write('Enter second number: '),
 read(B),
 multiply(A,B,X),
 write('Product            : '),
 write(X).

multiply(X,0,0):-!.
multiply(X,1,X):-!.
multiply(X,Y,Res):- N is Y-1, multiply(X,N,R), Res is R+X.
```

```
1 ?- go.

Enter first number : 5.
Enter second number: |: 12.
Product            : 60
true.

2 ?- go.

Enter first number : 9.
Enter second number: |: 0.
Product            : 0
true.

3 ?-
```

11. Write a program in PROLOG to implement towerofhanoi (N) where N represents the number of discs.

```
go:- nl,write('Enter no. of disks: '), read(N),nl,
     toh(N).

toh(N) :- move(N,left,right,middle).

move(0,_,_,_):- !.
move(1,A,B,_):- inform(A,B),!.

move(N,A,B,C):- M is N-1,
            move(M,A,C,B),
            inform(A,B),
            move(M,C,B,A).

inform(A,B)  :- write('Move disk from '),
                        write(A),write(' to '),write(B),write('.'),nl.
```

```
1 ?- go.

Enter no. of disks: 3.

Move disk from left to right.
Move disk from left to middle.
Move disk from right to middle.
Move disk from left to right.
Move disk from middle to left.
Move disk from middle to right.
Move disk from left to right.
true.

2 ?-
```

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

```
go:-
 nl,write('Enter a list : '),
 read(L),
 write('Enter element: '),
 read(E),
 memb(E,L),!.

go:- nl,write('Element not found.').

/*
 * or,
 *
 * memb(E,L);
 * nl,write('Element not found.').
 *
```

```
 */

memb(X,[X|_]) :- nl,write('Element found.').
/*
 * if the element to be found is same as the head.
 * memb(X,[X|T]), T occurs once, singleton variable,
 * use anonymous variable.
 * memb(X,[X|_]). for one occurence.
 */

memb(X,[_|T]) :- memb(X,T).
/*
 * if the element exists in the tail.
 * memb(X,[H|T]) :- memb(X,T).
 * H occurs once, use anonymous variable.
 */
```

```
1 ?- go.

Enter a list : [1,4,5,2,3,9].
Enter element: |: 2.

Element found.
true.

2 ?- go.

Enter a list : [1,4,5,2,3,9].
Enter element: |: 10.

Element not found.
true.

3 ?-
```

13. Write a PROLOG program to implement last_el (L, X) where L is a list and X represents the last element of list L.

```
go:-
 nl,write('Enter a list : '),read(L),
 last_el(L,X),
 write('Last element : '),
 write(X),nl.

last_el([X],X):-!.
last_el([H|T],X):- last_el(T,X).
```

```
1 ?- go.

Enter a list : [1,2,3,4,5].
Last element : 5
true.

2 ?- go.

Enter a list : [2].
Last element : 2
true.
```

14. Write a menu-driven PROLOG program to implement the following:

- **a. conc (L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.**
- **b. reverse (L, R) where List L is original and List R is a reversed list.**
- **c. palindrome (L) which checks whether a list L is a palindrome or not.**

```
go:-
 nl,nl,write('   Menu'),nl,
 write('1. Append two lists.'),nl,
 write('2. Reverse a list.'),nl,
 write('3. Check whether a list is a palindrome'),nl,
 nl,write('Enter choice : '), read(C),
 choice(C).

 choice(1) :-     !,    nl,write('Enter first list : '),
                        read(L1),
                        write('Enter second list: '),
                        read(L2),
                        conc(L1,L2,L),
                        write('Appended list    : '),
                        write(L).

 choice(2) :- !,        nl,write('Enter a list : '),
                        read(L),
                        rev(L,R),
                        write('Reversed list: '),
                        write(R).

 choice(3) :- nl,write('Enter a list : '),
              read(L),
              palindrome(L),nl,!.
 choice(3) :- !,  write('Not a palindrome.'),nl.

 choice(X) :- nl,write('Invalid choice!'), go.

conc([],L,L).
conc([X|L1],L2,[X|L3]):- conc(L1,L2,L3).

rev([],[])   :- !.
rev([X],[X]) :- !.
rev([X|T],L) :- rev(T,R), append(R,[X],L).

palindrome(L) :- reverse(L,L),write('Palindrome.').
```

```
1 ?- go.


   Menu
1. Append two lists.
2. Reverse a list.
3. Check whether a list is a palindrome

Enter choice : 1.

Enter first list : |: [1,2,3].
Enter second list: |: [4,5,6].
Appended list    : [1,2,3,4,5,6]
true.

2 ?- go.


   Menu
1. Append two lists.
2. Reverse a list.
3. Check whether a list is a palindrome

Enter choice : 2.

Enter a list : |: [1,2,3,4].
Reversed list: [4,3,2,1]
true.

3 ?- go.


   Menu
1. Append two lists.
2. Reverse a list.
3. Check whether a list is a palindrome

Enter choice : 3.

Enter a list : |: [1,2,3,2,1].
Palindrome.
true.
```

15. Write a PROLOG program to implement nth_element (N, L, X) where N is the desired position, L is a list and X represents the Nth element of L.

```
go:-
 nl,write('Enter a list  : '),
 read(L),
 write('Enter position: '),
 read(N),
 nth_ele(N,L,X),
 write('Element       : '),
 write(X),nl.

nth_ele(1,[X|T],X):-!.
nth_ele(N,[H|T],X):- N1 is N-1, nth_ele(N1,T,X).
```

```
1 ?- go.

Enter a list   : [1,2,3,4,5,6].
Enter position: |: 4.
Element        : 4
true.

2 ?- go.

Enter a list   : [5,3,2,1,0,4].
Enter position: |: 2.
Element        : 3
true.

3 ?-
```

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

```
go:-
 nl,write('Enter list : '),
 read(L),
 sumlist(L,Sum),
 write('Sum         : '),
 write(Sum).

sumlist([],0).
sumlist([X|T], Res) :- sumlist(T,R1), Res is X + R1.
```

```
1 ?- go.

Enter list : [1,2,3,4,5,6,7].
Sum        : 28
true.

2 ?- go.

Enter list : [].
Sum        : 0
true.

3 ?-
```

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

```
go:-
 nl,write('Enter a list    : '), read(L), maxlist(L,X),
 write('Largest element : '),
 maxlist(L,X), write(X),nl.

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

maxlist([X],X) :- !.
maxlist([H|T],M) :- maxlist(T,M1), max(H,M1,M).
```

```
1 ?- go.

Enter a list    : [1,2,4,9,3,7].
Largest element : 9
true.

2 ?-
```

18. 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.

```
go:-
    nl,write('Enter a list: '),read(L),nl,
    write('1. Check even length.'),nl,
    write('2. Check odd length.'),nl,
    nl,write('Enter choice: '),
    read(C), choice(C,L),nl.

choice(1,L) :- evenlength(L),nl,write('Length is even'),!.
choice(1,L) :- nl, write('Length is not even').

choice(2,L) :- oddlength(L),nl,write('Length is odd'),!.
choice(2,L) :- nl, write('Length is not odd').

evenlength([])    :- !.
evenlength([H|T]) :- oddlength(T).

oddlength([_])   :- !.
oddlength([H|T]) :- evenlength(T).
```

```
1 ?- go.

Enter a list: [1,2,3,4,5].

1. Check even length.
2. Check odd length.

Enter choice: |: 1.

Length is not even
true.

2 ?- go.

Enter a list: [1,2,3,4,5].

1. Check even length.
2. Check odd length.

Enter choice: |: 2.

Length is odd
true.

3 ?-
```

19. Write a PROLOG program to implement delete_first (X, L ,R) where X denotes the element whose first occurrence has to be deleted from list L to obtain list R.

```
go:-
 nl,write('Enter a list               : '),
 read(L),
 write('Enter element to be deleted: '),
 read(E),
 delete_first(E,L,X),
 write('Deleted list              : '),
 write(X),nl.

delete_first(X,[X|T],T):-!.
delete_first(X,[H|T],[H|T1]) :- delete_first(X,T,T1).

/*
 * Or use,
 *
 * delete_first(X,[X|T],T).
 * delete_first(X,[H|T],[H|T1]) :- X\=H,delete_first(X,T,T1).
 *
 * Or use,
 *
 * delete_first(X,[X|T],L):-L is T,!.
 * delete_first(X,[H|T],[H|T1]) :- delete_first(X,T,T1).
 *
 */
```

```
1 ?- go.

Enter a list               : [1,2,3,4,5,3,2,1,5].
Enter element to be deleted: |: 1.
Deleted list               : [2,3,4,5,3,2,1,5]
true.

2 ?- go.

Enter a list               : [1,2,3,4,5,3,2,1,5].
Enter element to be deleted: |: 3.
Deleted list               : [1,2,4,5,3,2,1,5]
true.

3 ?-
```

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

```
go:-
 nl,write('Enter a list  : '),
 read(L),
 write('Enter element : '),
 read(E),
```

```
 write('Enter position: '),
 read(P),
 length(L,Len),Length is Len+1,
 P =< Length, insert_nth(E,P,L,Res),
 write('List after insertion: '),
 write(Res),nl.

go:- nl,write('Error!').

insert_nth(I,1,L,[I|L]):-!.
insert_nth(I,N,[H|T1],[H|T2]):- N1 is N-1,insert_nth(I,N1,T1,T2).
```

```
1 ?- go.

Enter a list  : [1,2,3,4,5,3,2,1,5].
Enter element : |: 7.
Enter position: |: 3.
List after insertion: [1,2,7,3,4,5,3,2,1,5]
true .

2 ?- go.

Enter a list  : [1,2,3,4,5].
Enter element : |: 10.
Enter position: |: 1.
List after insertion: [10,1,2,3,4,5]
true .

3 ?-
```

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

```
go:-
 nl,write('Enter a list  : '),
 read(L),
 write('Enter position: '),
 read(P),
 length(L,Len), P =< Len,
 delete_nth(P,L,Res),nl,
 write('List after deletion: '),
 write(Res),nl.

go:- nl,write('Error!').

delete_nth(1,[H|T],T):-!.
delete_nth(N,[H|T1],[H|T2]):- N1 is N-1, delete_nth(N1,T1,T2).
```

```
1 ?- go.

Enter a list  : [1,2,3,4,5].
Enter position: |: 3.

List after deletion: [1,2,4,5]
true .

2 ?- go.

Enter a list  : [1,2,3,4,5].
Enter position: |: 5.

List after deletion: [1,2,3,4]
true .

3 ?-
```

22. Write a program in PROLOG to implement remove_dup (L, R) where L denotes the list with some duplicates and the list R denotes the list with duplicates removed.

```
go:-
    nl,write('Enter list: '),
    read(L),nl,
    write('List with duplicates removed:'),nl,
    remove_dup(L,R), write(R),nl.

remove_dup([],[]):-!.
remove_dup([H|T],L):- member(H,T),remove_dup(T,L),!.
remove_dup([H|T1],[H|T2]):- remove_dup(T1,T2).
```

```
1 ?- go.

Enter list: [1,1,1,1,2,3,2,5,6,7,5].

List with duplicates removed:
[1,3,2,6,7,5]
true.

2 ?- go.

Enter list: [1,1,1,1,1,1].

List with duplicates removed:
[1]
true.

3 ?-
```

23. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.

```
go:-
      nl,write('Enter first list : '),read(L1),
      write('Enter second list: '),read(L2),
```

```
      merge(L1,L2,M),
      write('Merged list      : '),write(M),
      nl.

merge([],A,A):-!.

merge(A,[],A):-!.

merge([H1|T1],[H2|T2],X) :- H1=<H2, merge(T1,[H2|T2],Y),
                            append([H1],Y,X),!.

merge([H1|T1],[H2|T2],X) :- merge([H1|T1],T2,Y), append([H2],Y,X),!.
```

```
1 ?- go.

Enter first list : [1,3,5,7,9].
Enter second list: |: [2,4,6,8,10].
Merged list      : [1,2,3,4,5,6,7,8,9,10]
true.

2 ?-
```

24. Write a program in PROLOG to implement permute (L, P) where P represents all possible permutations of the elements of List L.

```
go:-
 nl,write('Enter a list    : '),
 read(L),
 nl,write('Possible permutations'),nl,
 permute(L,R),write(R),fail.

permute([],[]).
permute(L,[H|T]) :- append(V,[H|U],L),
                    append(V,U,W),
                    permute(W,T).
```

```
1 ?- go.

Enter a list    : [1,2,3].

Possible permutations
[1,2,3][1,3,2][2,1,3][2,3,1][3,1,2][3,2,1]
false.

2 ?- go.

Enter a list    : [1,3].

Possible permutations
[1,3][3,1]
false.

3 ?-
```

25. Write a program in PROLOG to implement is_subsequence(S,L) so that it returns true if S is a subsequence of list L.

```
go:-
 nl,write('Enter a list    : '),
 read(L),
 write('Enter a subsequence : '),
 read(S),
 is_subsequence(S,L).

is_subsequence([],[_|_])      :-!.
is_subsequence([H|T],[H|T1])  :- is_subsequence(T,T1),!.
is_subsequence([H|T],[H1|T1]) :- is_subsequence([H|T],T1),!.
```

```
1 ?- go.

Enter a list    : [1,2,3,4,5,6].
Enter a subsequence : |: [1,3,5].

true.

2 ?- go.

Enter a list    : [1,2,3,4,5,6].
Enter a subsequence : |: [3,2,1].

false.

3 ?-
```

26. Write a program in PROLOG to implement is_sublist(S,L) so that it returns true if S is a sublist of list L.

```
go:-
 nl,write('Enter a list    : '), read(L),
 write('Enter a sublist : '), read(S),
 is_sublist(S,L).


is_sublist(S,L):- append(L1,L2,L), append(S,L3,L2).
```

```
1 ?- go.

Enter a list    : [1,2,3,4,5,6].
Enter a sublist : |: [1,3,5].

false.

2 ?- go.

Enter a list    : [1,2,3,4,5,6].
Enter a sublist : |: [2,3,4].

true .
```

27. Write a program in PROLOG to implement delete_all (X, L, R) where X denotes the element who's all occurrences has to be deleted from list L to obtain list R.

```
go:-
 nl,write('Enter a list               : '),
 read(L),
 write('Enter element to be deleted: '),
 read(E),
 delete_all(E,L,X),
 write('Deleted list              : '),
 write(X),nl.


delete_all(_,[],[]).
delete_all(X,[X|T],L) :- delete_all(X,T,L),!.
delete_all(X,[H1|T1],[H1|T2]) :- delete_all(X,T1,T2).
```

```
1 ?- go.

Enter a list               : [1,1,2,1,3,4,5,1,6].
Enter element to be deleted: |: 1.
Deleted list               : [2,3,4,5,6]
true.

2 ?- go.

Enter a list               : [1,1,1,1,1,1].
Enter element to be deleted: |: 1.
Deleted list               : []
true.

3 ?-
```

28. Write a program in PROLOG to implement inorder(T), preorder(T) and postorder(T), where T represents a tree, to print inorder, preorder and postorder traversals of the tree.

```
go:-
    nl,write('Enter a tree: '),
    read(T),nl,
    write('Preorder : '),pre(T),nl,
    write('Inorder  : '),in(T),nl,
    write('Postorder: '),post(T),nl.

pre(nil):-!.
pre(T) :- T=t(X,Y,Z), write(Y), write(' '), pre(X),pre(Z).

in(nil):-!.
in(T) :- T=t(X,Y,Z), in(X), write(Y), write(' '), in(Z).

post(nil):-!.
post(T) :- T=t(X,Y,Z), post(X),post(Z),write(Y), write(' ').
```

```
1 ?- go.

Enter a tree: t(t(t(nil,4,nil),2,t(nil,5,nil)),1,t(t(nil,6,nil),3,t(nil,7,nil))).

Preorder : 1 2 4 5 3 6 7
Inorder  : 4 2 5 1 6 3 7
Postorder: 4 5 2 6 7 3 1
true.

2 ?-
```

29. Write a program in PROLOG to implement Depth-First-Search of a tree.

```
go:-
    nl,write('Enter a tree: '), read(T),nl,
    write('Enter element: '),read(Ele),
    dfs(T,Ele),nl,write('Element not found.'),!.

go:-
      nl,write('Element found.').

dfs(nil, ):-!.
dfs(t(X,Y,Z),Y) :- !,fail.
dfs(t(X,Y,Z),Ele) :- dfs(X,Ele),dfs(Z,Ele).
```

```
1 ?- go.

Enter a tree: t(t(t(nil,4,nil),2,t(nil,5,nil)),1,t(t(nil,6,nil),3,t(nil,7,nil))).

Enter element: |: 12.

Element not found.
true.

2 ?- go.

Enter a tree: t(t(t(nil,4,nil),2,t(nil,5,nil)),1,t(t(nil,6,nil),3,t(nil,7,nil))).

Enter element: |: 7.

Element found.
true.

3 ?- go.
```

30. Write a PROLOG program that will take grammar rules in the following format:

NT (NT | T)*

Where NT is any nonterminal, T is any terminal and Kleene star (*) signifies any number of repetitions, and generate the corresponding top-down parser, that is:

sentence → noun-phrase, verb-phrase

determiner → [the]

will generate the following:

sentence (I, O) :- noun-phrase(I,R), verb-phrase (R,O).

determiner ([the|X], X) :- !.

```
utterance(X) :- sentence(X,[]).

sentence(Start,End) :- nounphrase(Start,Rest),verbphrase(Rest,End).

nounphrase([Noun|End],End) :- noun(Noun).
nounphrase([Article,Noun|End],End) :- article(Article), noun(Noun).

verbphrase([Verb|End],End) :- verb(Verb),!.
verbphrase([Verb|Rest],End) :- verb(Verb), nounphrase(Rest,End).

article(a).
article(the).
noun(man).
noun(dog).
verb(likes).
verb(bites).
```