Aim: Implement following program using SWI-Prolog.

1. **Write simple fact for following:**
   a. **Ram likes mango.**
   b. **Seema is a girl.**
   c. **Bill likes Cindy.**
   d. **Rose is red.**
   e. **John owns gold.**

Program:

```
likes(ram,mango).

likes(bill,candy).

girl(seema).

red(rose).

owns(john,gold).
```

Output:

```
?- likes(X,Y).
X = ram,
Y = mango .

?- likes(who,what).
false.

?- likes(Who,What).
Who = ram,
What = mango .

?- girl(Who).
Who = seema.

?- owns(john,gold).
true.
```

2. **Write predicates one converts Celsius temperatures to Fahrenheit, the other checks if the temperature is below freezing.**

Program:

```
c_to_f(C,F):-
        F is C*9/5+32.

freezing(F):-
```

```
    F=<32,
    write("It's freezing").
freezing(F):-
    write("It's not freezing").
```

Output:

```
?- c_to_f(0,F).
F = 32.

?- c_to_f(1,F).
F = 33.8.

?- freezing(33).
It's not freezing
true.

?- freezing(31).
It's freezing
true
```

3. Write Prolog clauses to check whether a given line segment is horizontal, vertical or oblique.

Program:

```
vertical(seg(point(X,Y),point(X,Y1))).
horizontal(seg(point(_,Y),point(_,Y))).
oblique(seg(point(X1,Y1),point(X2,Y2))):-
    X1\==X2,
    Y1\==Y2.
```

Output:

```
?- vertical(seg(point(2,4),point(2,8))).
true.

?- horizontal(seg(point(3,2),point(2,2))).
true.

?- oblique(seg(point(3,4),point(4,3))).
true.
```

4. Write a program to implement to print factorial, Fibonacci of a given number.

Fibonacci Program:

```
fib(X,Y):-
    X>0,
    fib(X, Y, _).
fib(0,0).


fib(X,Y1,Y2
    ):- X > 0,
    X1 is X - 1,
    fib(X1,Y2,Y3),
    Y1 is Y2 + Y3,
    write(Y1),nl.
fib(0,1,0).
```

Factorial Program:

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

Fibonacci Output:

```
;-
% f:/sem 7/AI/pr3/pr3_4_1.pl compiled 0.00 sec, 2 clauses
?- fact(5,F).
F = 120.

?- fact(6,F).
F = 720.

?- fact(0,F).
F = 1.
```

Factorial Output:

```
?- fib(0,Y).
Y = 0.

?- trace.
true.

[trace]  ?- fib(0,Answer).
   Call: (10) fib(0, _14976) ? creep
   Call: (11) 0>0 ? creep
   Fail: (11) 0>0 ? creep
   Redo: (10) fib(0, _14976) ? creep
   Exit: (10) fib(0, 0) ? creep
Answer = 0.

[trace]  ?- nottrace.
Correct to: "notrace"?
Please answer 'y' or 'n'? yes
true.

[debug]  ?- fib(5,Ans).
1
2
3
5
8
Ans = 8.

[debug]  ?- fib(8,Ans).
1
2
3
5
8
13
21
34
Ans = 34.
```

5. Write a Prolog program to formulate rules to capture the following relationships for following family tree.

Program:

```
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),paarent(Z,Y),female(X),X\==Y.

brother(X,Y):-parent(Z,X),paarent(Z,Y),male(X),X\==Y.


grandparent(X,Y):-parent(X,Z),parent(Z,Y).


grandmother(X,Z):-mother(X,Y),parent(Y,Z).

grandfather(X,Z):-father(X,Y),parent(Y,Z).

wife(X,Y):-parent(X,Z),parent(Y,Z),female(X),male(Y).

uncle(X,Z):-brother(X,Y),parent(Y,Z).

predecessor(X,Z):-parent(X,Z).

predecessor(X,Z):-parent(X,Y),predecessor(Y,Z).

female(kdt).

female(kgt).

female(rgc).

female(rjt).

male(adt).

male(tgt).

male(ygc).

male(jgt).

parent(kdt,tgt).

parent(ygc,tgt).
```

parent(ygc,kgt).

parent(tgt,rjt).

parent(tgt,rgc).

parent(rgc,adt).

parent(tgt,jgt).

parent(jgt,peter).

Output:

```
% f:/sem 7/AI/pr3/pr3_5.pl compiled 0.00 sec, -1 clauses
?- trace.
true.

[trace]  ?- mother(kdt,tgt).
   Call: (10) mother(kdt, tgt) ? creep
   Call: (11) parent(kdt, tgt) ? creep
   Exit: (11) parent(kdt, tgt) ? creep
   Call: (11) female(kdt) ? creep
   Exit: (11) female(kdt) ? creep
   Exit: (10) mother(kdt, tgt) ? creep
true.

[trace]  ?- nottrace.
Correct to: "notrace"? yes
true.

[debug]  ?- mother(X,Y).
X = kdt,
Y = tgt .

[debug]  ?- mother(kdt,Y).
Y = tgt.

[debug]  ?- mother(kdt,tgt).
true.

[debug]  ?- mother(adt,tgt).
false.
```

6. **Write a program to solve the following problem. Imagine a room containing a monkey, chair and some bananas. That has been hung from the center of the 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.**

Program:

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

Output:

```
?- can_reach(monkey,bananas).
true.

?- can_reach(Who,Whom).
Who = monkey,
Whom = bananas ;

?- trace
|    .
true.

[trace]  ?- can_reach(X,Y).
   Call: (10) can_reach(_15074, _15076) ? creep
   Call: (11) dexterous(_15074) ? creep
   Exit: (11) dexterous(monkey) ? creep
   Call: (11) vclose(monkey, _15076) ? creep
   Call: (12) get_on(monkey, _15692) ? creep
   Call: (13) can_climb(monkey, _15736) ? creep
   Exit: (13) can_climb(monkey, chair) ? creep
   Exit: (12) get_on(monkey, chair) ? creep
   Call: (12) under(chair, _15076) ? creep
   Call: (13) in_room(_15910) ? creep
   Exit: (13) in_room(bananas) ? creep
   Call: (13) in_room(chair) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) in_room(_15076) ? creep
   Exit: (13) in_room(bananas) ? creep
   Call: (13) can_move(bananas, chair, bananas) ? creep
   Fail: (13) can_move(bananas, chair, bananas) ? creep
   Redo: (13) in_room(_15076) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) can_move(bananas, chair, chair) ? creep
   Fail: (13) can_move(bananas, chair, chair) ? creep
   Redo: (13) in_room(_15076) ? creep
   Exit: (13) in_room(monkey) ? creep
   Call: (13) can_move(bananas, chair, monkey) ? creep
   Fail: (13) can_move(bananas, chair, monkey) ? creep
   Redo: (13) in_room(_16614) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) in_room(chair) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) in_room(_15076) ? creep
   Exit: (13) in_room(bananas) ? creep
   Call: (13) can_move(chair, chair, bananas) ? creep
   Fail: (13) can_move(chair, chair, bananas) ? creep
   Redo: (13) in_room(_15076) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) can_move(chair, chair, chair) ? creep
   Fail: (13) can_move(chair, chair, chair) ? creep
   Redo: (13) in_room(_15076) ? creep
   Exit: (13) in_room(monkey) ? creep
   Call: (13) can_move(chair, chair, monkey) ? creep
   Fail: (13) can_move(chair, chair, monkey) ? creep
   Redo: (13) in_room(_17318) ? creep
   Exit: (13) in_room(monkey) ? creep
   Call: (13) in_room(chair) ? creep
   Exit: (13) in_room(chair) ? creep
   Call: (13) in_room(_15076) ? creep
   Exit: (13) in_room(bananas) ? creep
   Call: (13) can_move(monkey, chair, bananas) ? creep
   Exit: (13) can_move(monkey, chair, bananas) ? creep
   Exit: (12) under(chair, bananas) ? creep
   Call: (12) tall(chair) ? creep
   Exit: (12) tall(chair) ? creep
   Exit: (11) vclose(monkey, bananas) ? creep
   Exit: (10) can_reach(monkey, bananas) ? creep
X = monkey,
Y = bananas ;
```

7. Write a program for search, concatenate, insert and remove function of lists.

Program:

```
/*search an element */

disp([H|T]):-
    write("Enter element to search for:"),
    read(N),
    find_else([H|T],N).
find_else([H|T],N):-
    H\=[],
    (N=H->writef("%t is a member of list",[N]);find_else(T,N));

    T=[],
    writef("%t is not a member of list",[N]).


/*to add an element */

add(X,L,[X|L]).

/*concatination*/

concate([H1|T1],L2,[H1|T2]):-
    concate(T1,L2,T2).
concate([],L2,L2).

/*delete element from list */

de(H,[H],[]).
de(X,[X|T1],T1).
de(X,[H|T],[H|T1]):-
    de(X,T,T1).
de(X,[_],_):-
    writef("Element %t not found",[X]).
```

Output:

```
?- disp([1,2,3,4,5]).
Enter element to search for:2
|: .
2 is a member of list
true .

?- disp([1,2,3,4,5]).
Enter element to search for:7.
7 is not a member of list
true .

?- concate([1,2,3],[4,5,6],Ans).
Ans = [1, 2, 3, 4, 5, 6].

?- add(5,[1,2,3,4],Ans).
Ans = [5, 1, 2, 3, 4].

?- de(3,[1,2,3,4,5],Ans).
Ans = [1, 2, 4, 5]
```

8. Write a program in prolog for medical diagnosis.

Program:

```
go :-
 write('What is the patient''s name? '),
 readln(Patient), hypothesis(Patient,Disease),
 write_list([Patient,'probably has ',Disease,'.']),nl.
go :-
 write('Sorry, I don''t seem to be able to'),nl,
 write('diagnose the disease.'),nl.


symptom(Patient,fever) :-
 write_list(['Does ',Patient,' have a fever (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,rash) :-
 write_list(['Does ',Patient,' have a rash (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,headache) :-
 write_list(['Does ',Patient,' have a headache (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,runny_nose) :-
 write_list(['Does ',Patient,' have a runny_nose (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,conjunctivitis) :-
```

```
 write_list(['Does ',Patient,' have a conjunctivitis (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,cough) :-
 write_list(['Does ',Patient,' have a cough (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,body_ache) :-
 write_list(['Does ',Patient,' have a body_ache (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,chills) :-
 write_list(['Does ',Patient,' have a chills (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,sore_throat) :-
 write_list(['Does ',Patient,' have a sore_throat (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,sneezing) :-
 write_list(['Does ',Patient,' have a sneezing (y/n) ?']),
 response(Reply),
 Reply='y'.
symptom(Patient,swollen_glands) :-
 write_list(['Does ',Patient,' have a swollen_glands (y/n) ?']),
 response(Reply),
 Reply='y'.
```

```
hypothesis(Patient,measles) :-
 symptom(Patient,fever),
 symptom(Patient,cough),
 symptom(Patient,conjunctivitis),
 symptom(Patient,runny_nose),
 symptom(Patient,rash).
hypothesis(Patient,german_measles) :-
 symptom(Patient,fever),
 symptom(Patient,headache),
 symptom(Patient,runny_nose),
 symptom(Patient,rash).
hypothesis(Patient,flu) :-
 symptom(Patient,fever),
 symptom(Patient,headache),
 symptom(Patient,body_ache),
 symptom(Patient,conjunctivitis),
 symptom(Patient,chills),
 symptom(Patient,sore_throat),
 symptom(Patient,runny_nose),
 symptom(Patient,cough).
hypothesis(Patient,common_cold) :-
 symptom(Patient,headache),
 symptom(Patient,sneezing),
 symptom(Patient,sore_throat),
 symptom(Patient,runny_nose),
 symptom(Patient,chills).
hypothesis(Patient,mumps) :-
```

```
 symptom(Patient,fever),
 symptom(Patient,swollen_glands).
hypothesis(Patient,chicken_pox) :-
 symptom(Patient,fever),
 symptom(Patient,chills),
 symptom(Patient,body_ache),
 symptom(Patient,rash).
hypothesis(Patient,measles) :-
 symptom(Patient,cough),
 symptom(Patient,sneezing),
 symptom(Patient,runny_nose).
 write_list([]).
 write_list([Term| Terms]) :-
 write(Term),
 write_list(Terms).
response(Reply) :-
 get_single_char(Code),
 put_code(Code), nl,
 char_code(Reply, Code).
```

Output:

```
?- go.
What is the patient's name? dharmay.
Does dharmay have a fever (y/n) ?

Does dharmay have a fever (y/n) ?n
Does dharmay have a fever (y/n) ?n
Does dharmay have a headache (y/n) ?y
Does dharmay have a sneezing (y/n) ?y
Does dharmay have a sore_throat (y/n) ?y
Does dharmay have a runny_nose (y/n) ?y
Does dharmay have a chills (y/n) ?y
dharmayprobably has common_cold.
true
```