**Practical – 1**

**Knowledge Representation (Prolog programming)**

1. **Write a program in Prolog to implement simple facts and Queries**

* **Software requirements:** GNU Prolog
* **Theory:**

A **fact** is a predicate expression that makes a declarative statement about the problem domain. Whenever a variable occurs in a Prolog expression, it is assumed to be **universally quantified**.

likes(john, susie). /\* John likes Susie \*/

The Prolog interpreter responds to **queries** about the facts and rules represented in its database. The database is assumed to represent what is true about a particular problem domain. In making a query you are asking Prolog whether it can prove that your query is true. If so, it answers "yes" and displays any **variable bindings** that it made in coming up with the answer. If it fails to prove the query true, it answers "No".

Whenever you run the Prolog interpreter, it will **prompt** you with **?-**.

**Code:**

male(yatharth).

male(karan).

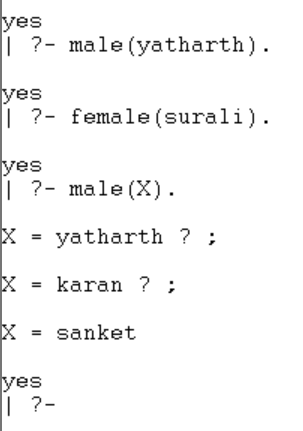
male(sanket).

female(surali).

female(vaishnavi).

female(kavya).

**Output:**

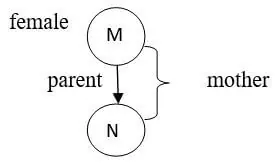
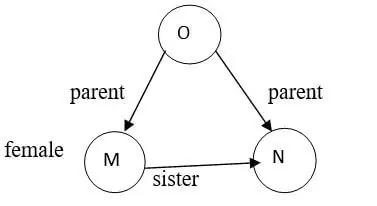


**Conclusion:** We learned to implement simple program in Prolog.

1. **Design a Family Tree for Your Family: Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin.**

* **Software requirements:** GNU Prolog
* **Theory:**

The prolog syntax for mother-sister relationship is given as below:

Mother Relationship Sister Relationship

mother(M,N): parent(M,N), female(M).

sister(M,N): parent(O,M), parent(O,N), female(M), M\= =N.

We can also define:

* father(M,N): parent(M,N),female(M).
* haschild(M): parent(M,\_).
* brother(M,N): parent(O,M), parent(O,N), male(M),M\==N.

Where \_(underscore) indicates that it is an anonymous variable.

We can also write the syntax for some other relationships as follows:

* grandparent(M,N): parent(M,O), parent(O,N).
* grandmother(M,O): mother(M,N), parent(N,O).
* grandfather(M,O): father(M,N), parent(N,O).
* wife(M,N): parent(M,O), parent(N,O), female(M),male(N).
* uncle(M,O): brother(M,N), parent(N,O).

The program in prolog specifies the relationship between objects and the properties of objects; the family trees tell us how to construct a database of family.

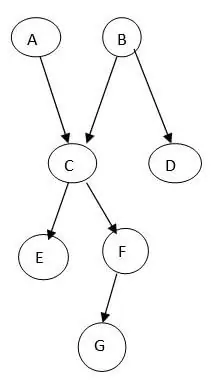
**Brother relationship:**

* If they both are male.
* If they have the same parent.

Suppose we have some clauses to illustrate the relationship:

* + Let us see the working of the family tree by considering an example that can be formed from the prolog family tree.

The sample family tree is given below:



**Code**

male(ratilal).

male(tejas).

male(jayesh).

male(dhiren).

male(yatharth).

male(divy).

male(mumux).

female(tara).

female(niru).

female(bhavna).

female(rajeshree).

female(vishwa).

female(drashti).

parent(tejas,yatharth).

parent(niru,yatharth).

parent(jayesh,divy).

parent(bhavna,divy).

parent(jayesh,vishwa).

parent(bhavna,vishwa).

parent(jayesh,drashti).

parent(bhavna,drashti).

parent(dhiren,mumux).

parent(rajeshree,mumux).

father(X,Y) :- male(X),parent(X,Y).

mother(X,Y) :- female(X),parent(X,Y).

son(X,Y) :- male(X),parent(Y,X).

daughter(X,Y) :- female(X),parent(X,Y).

grandfather(X,Y) :- male(X),parent(X,Somebody),parent(Somebody,Y).

grandmother(X,Y) :- female(X),parent(X,Somebody),parent(Somebody,Y).

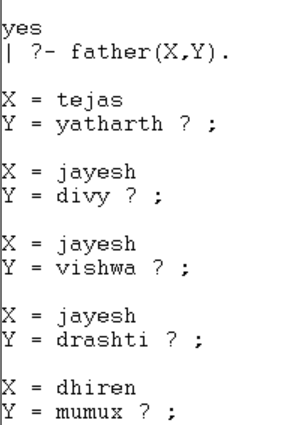
brother(X,Y) :- male(X),parent(Somebody,X),parent(Somebody,Y).

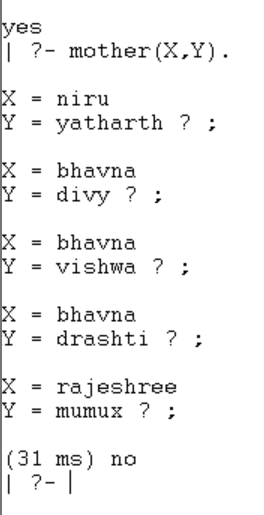
sister(X,Y) :- female(X),parent(Somebody,X),parent(Somebody,Y).

uncle(X,Y) :- male(X),brother(X,Somebody),parent(Somebody,Y).

aunty(X,Y) :- female(Y),uncle(Somebody,X),spouse(Somebody,Y).

**Output**



****

**Conclusion**: We learned to form functions for relations and ‘and’ & ‘or’ operations.

* 1. **Design a Medical Diagnosis Expert System**

**Software requirements:** GNU Prolog

**Theory:** Make function for every type of yes and no to convert into y and n.

Then make function ask for cold, cough and fever accordingly give results.

**Code**:

answer(Inp,X):-((Inp='Yes',X='y');(Inp='yes',X='y');(Inp='y',X='y');(Inp='Y',X='y');(Inp='YES',X='y'));((Inp='NO',X='n');(Inp='no',X='n');(Inp='N',X='n');(Inp='n',X='n')).

mainask:-write('Write your name :'),read(Name),cold(C),cough(F),fever(V),fun(C,F,V,Ans),write(Name),write(Ans).

fever(V):-write('Fever ? '),read(Inp),answer(Inp,V).

cough(F):-write('Cough ? '),read(Inp),answer(Inp,F).

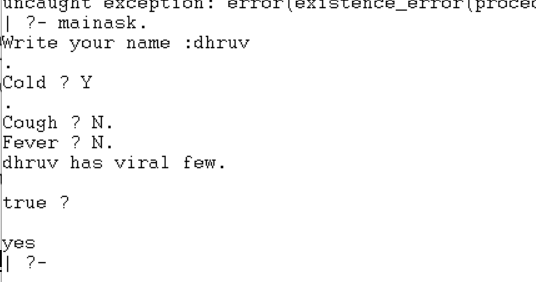
cold(C):-write('Cold ? '),read(Inp),answer(Inp,C).

fun(C,F,V,Ans):-(C='y',F='y',V='y',Ans=' has viral few.');(C='n',F='n',V='y',Ans=' has viral fever.');

(C='y',F='n',V='n',Ans=' has normal cold.');(C='n',F='n',V='n',Ans=' is alright!');

(C='n',F='y',V='n',Ans=' has normal cough.');(C='y',F='y',V='n',Ans=' has normal cold and cough.').

**Output:**



**Conclusion:** We learned to perform arithmetic operations on prolog, found sum of 1 to n integers and applied recursion.

1. **Write a Prolog program to demonstrate arithmetic operations and find addition of 1 to N numbers. Also demonstrate examples of recursion.**

* **Software requirements:** GNU Prolog
* **Theory:**

In arithmetic expression, any variables must already be bound. The value of these variables must be numerical. The value of arithmetic expression bounds the variable of first argument. If it is not, an error message will be generated as result.

In arithmetic expression, + - \* / symbols are special type of infix operator, and these operators are also known as arithmetic operators. In Prolog, operators are used as predicates but here operators are functions and these operators return a numerical value.

Arithmetic expressions can include variables, numbers, operators, and arithmetic functions. These will be written in parentheses with their arguments. These will return numerical values just like the arithmetic operators.

The minus(-) arithmetic operator is used as a binary infix operator, which is used to describe the difference of two numerical values like A - 2. It is also used as a unary prefix operator, which is used to describe the negative of a numerical value.

Looping is used to enable a set of instructions to be repeatedly executed either a fixed number of times or until a given condition met. Prolog has no looping facility, but we can obtain a similar effect. Using this effect, we can evaluate a sequence of goals repeatedly.

**Code:**

sumto(1, 1).

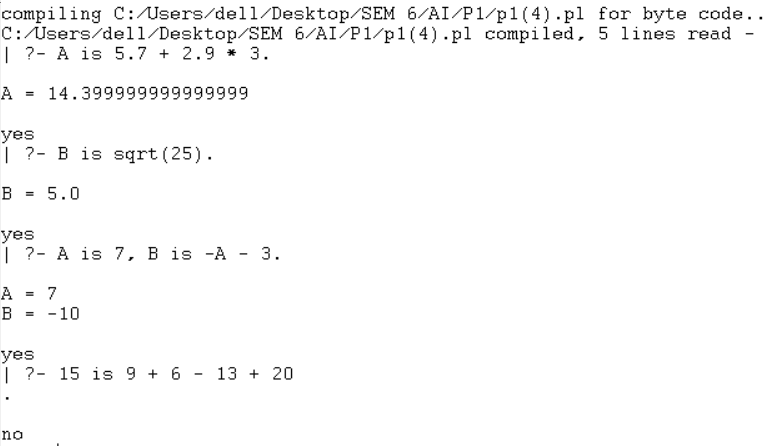
sumto(N, M) :- N>1, N1 is N-1, sumto(N1, M1), M is M1+N.

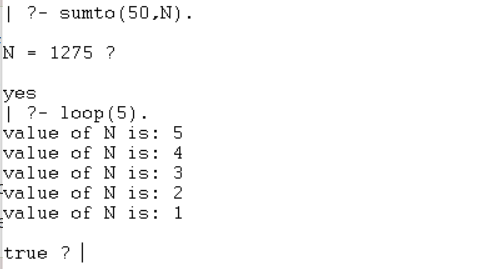
loop(0).

loop(N) :- N>0, write('value of N is: '), write(N), nl,

S is N-1, loop(S).

**Output:**





**Conclusion:** We learned to perform arithmetic operations on prolog, found sum of 1 to n integers and applied recursion.

1. **Write a program to display Fibonacci series in prolog.**

* **Software requirements:** GNU Prolog
* **Theory:**

The **Fibonacci sequence**, in which each number is the sum of the two preceding ones. The sequence commonly starts from 0 and 1.

The Fibonacci number I is defined as the sum of the Fibonacci numbers for I-1 and I-2, while the Fibonacci number of 0 and 1 are both defined to be 1.

The Fibonacci numbers may be defined by the recurrence relation

F0=0 and F1=1, F=F-1+F-2; �0=0,�1=1,

��=��−1+��−2

for *n* > 1.

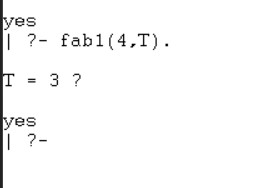
**Code:**

fab1(1,1).

fab1(2,1).

fab1(N,T):-N>2,N1 is N-1,N2 is N-2,fab1(N1,T1),fab1(N2,T2),T is (T1+T2).

**Output:**



**Conclusion:** We compiled the Fibonacci series.

1. **Write a program to find factorial of a number in prolog using recursion.**

* **Software requirements:** GNU Prolog
* **Theory:**

Prolog Factorial is the product of an integer and the other integers below the given number i.e., 5! is represented as 5\*4\*3\*2\*1 which is equal to 120.

Here in Prolog factorial,

* Prolog Factorial function definition is also similar to a normal factorial function.
* Factorial(0,1) i.e., factorial of 0 is generally 1.
* Factorial(N,M), if any temporary value N1 is assigned to N-1.
* Factorial(N1,M1), and is factorial of N1 is M1.
* M is NM1 i.e., assigning M to N\*M1, then value of N is M.

The above happens to be the recursive relation between N and factorial M. It reviews rules for particular relation in the top to bottom order.

**Code:**

fact(0,1).

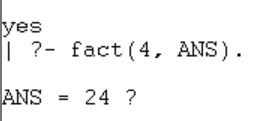
fact(N,ANS):- N>0,

N1 is N-1,

fact(N1,A),

ANS is A\*N.

**Output:**



**Conclusion:** We implemented factorial.