

## Experiment No. 6

Name : Sudhanshu Narsude

UID : 2019130044

Class : TE COMPS Batch C

Subject : AIML

**Aim :** To solve problems using Prolog Programming.

**Theory :**

What is Prolog?

Prolog or **PRO**gramming in **LOG**ics is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve **symbolic** or **non-numeric computation**. This is the main reason to use Prolog as the programming language in **Artificial Intelligence**, where **symbol manipulation** and **inference manipulation** are the fundamental tasks.

In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the **solution method**.

Prolog language basically has three different elements –

**Facts** – The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

**Rules** – Rules are extensions of facts that contain conditional clauses. To satisfy a rule these conditions should be met.

**Questions** – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

-----  
Q1) Create a family tree using PROLOG. It should have rules for father, mother, brother, sister, grandparent, uncle, aunt, predecessors, successors.

Program:

parent(snehal, pankaj).

parent(snehal, rajeev).

parent(ramesh, pankaj).

parent(ramesh, rajeev).  
parent(pankaj, om).  
parent(shraddha, om).  
female(snehal).  
female(shraddha).  
male(pankaj).  
male(om).  
male(ramesh).  
male(rajeev).  
mother(X, Y):- parent(X, Y), female(X).  
father(X, Y):- parent(X, Y), male(X).  
son(X, Y):- parent(Y, X), male(X).  
daughter(X, Y):- parent(Y, X), female(X).  
grandfather(X, Y):- parent(X, A), parent(A, Y), male(X).  
grandmother(X, Y):- parent(X, A), parent(A, Y), female(X).  
sister(X, Y):- parent(A, X), parent(A, Y), female(X), X \= Y.  
brother(X, Y):- parent(A, X), parent(A, Y), male(X), X \= Y.  
aunt(X, Y):- sister(X, Z), parent(Z, Y).  
uncle(X, Y):- brother(X, Z), parent(Z, Y).  
predecessor(X, Y) :- parent(X, Y).  
predecessor(X, Y) :- parent(X, A),predecessor(A, Y).  
successor(X, Y):- son(Y, X).  
successor(X, Y):- daughter(Y, X).  
successor(X, Y):- son(A, X), successor(A, Y).  
successor(X, Y):- daughter(A, X), successor(A, Y).

Output:

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.1)
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).
```

```
?- mother(X,om).
X = shraddha.
```

```
?- father(X,om).
X = pankaj .
```

```
?- son(om,X).
X = pankaj .
```

```
?- brother(X,pankaj)
|
X = rajeev .
```

```
?- predecessor(X,om)
|
X = pankaj .
```

Q2) Given a list [a,a,a,a,b,b,b,c,c]

write a function that does the following `rle([a,a,a,a,b,b,c,c],X)`

X: [a,b,c]

Program:


```
rle([],[]).
```

```
rle([X],[X]).
```

```
rle([X,X|Xs],Zs) :- rle([X|Xs],Zs).
```

```
rle([X,Y|Ys],[X|Zs]) :- X \= Y, rle([Y|Ys],Zs).
```

Output:

```
 SWI-Prolog (AMD64, Multi-threaded, version 8.4.1)
```

```
File Edit Settings Run Debug Help
```

```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.1)
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).
```

```
?- rle([a,a,a,a,b,b,b,c,c],X).
X = [a, b, c] ■
```

Q3) Given a list [a,b,c,d,e,f,g]

write a function that does the following `slice([a,b,c,d,e,f,g],[2,5],X)`

X: [c,d,e,f]

### Program:

slice([H|\_],[0,0],[H]).

slice([H|T],[0,To],[H|X]) :-

    N is To - 1,

    slice(T,[0,N],X).

slice(\_|T,[From,To], L) :-

    N is From - 1,

    M is To - 1,

    slice(T,[N,M],L).

### Output:

```
% c:/users/sudha/onedrive/documents/prolog/exp 6 compiled 0.00 sec, 0 clauses
% c:/users/sudha/onedrive/documents/prolog/exp 6 compiled 0.00 sec, 0 clauses
% c:/users/sudha/onedrive/documents/prolog/exp 6 compiled 0.00 sec, 0 clauses
?- slice([a,b,c,d,e,f,g],[2,5],L).
L = [c, d, e, f] .

?- slice([a,b,c,d,e,f,g],[1,5],L).
L = [b, c, d, e, f] .

?- slice([a,b,c,d,e,f,g],[3,7],L).
false.

?- slice([a,b,c,d,e,f,g],[3,6],L).
L = [d, e, f, g] .
```

Q4) Group list into sublists according to the distribution.given For example subsets([a,b,c,d,e,f,g],[2,2,3],X,[]) should return X = [[a,b][c,d][e,f,g]] The order of the list does not matter.

### Program :

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

el(X,\_|L,R) :- el(X,L,R).

selectN(0,\_,[]) :- !.

selectN(N,L,[X|S]) :- N > 0,

el(X,L,R),

N1 is N-1,

selectN(N1,R,S).

subsets([],[],[],[]).

subsets(G,[N1|Ns],[G1|Gs],[]) :-

selectN(N1,G,G1),

```
subtract(G,G1,R),
subsets(R,Ns,Gs,[]).
```

Output:

```
?- subsets([a,b,c,d,e,f,g],[2,2,3],X,[]).
X = [[a, b], [c, d], [e, f, g]]
```

### Q5) Huffman Code

We suppose a set of symbols with their frequencies, given as a list of  $fr(S,F)$  terms. Example:  $[fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)]$ . Our objective is to construct a list  $hc(S,C)$  terms, where  $C$  is the Huffman code word for the symbol  $S$ . In our example, the result could be  $Hs = [hc(a,'0'), hc(b,'101'), hc(c,'100'), hc(d,'111'), hc(e,'1101'), hc(f,'1100')] [hc(a,'01'),...etc.]$ . The task shall be performed by the predicate `huffman/2` defined as follows: %  
`huffman(Fs,Hs) :- Hs is the Huffman code table for the frequency table Fs`

Program:

```
huffman(Fs,Cs) :-
```

```
    initialize(Fs,Ns),
    make_tree(Ns,T),
    traverse_tree(T,Cs).
```

```
initialize(Fs,Ns) :- init(Fs,NsU), sort(NsU,Ns).
```

```
init([],[]).
```

```
init([fr(S,F)|Fs],[n(F,S)|Ns]) :- init(Fs,Ns).
```

```
make_tree([T],T).
```

```
make_tree([n(F1,X1),n(F2,X2)|Ns],T) :-
```

```
    F is F1+F2,
    insert(n(F,s(n(F1,X1),n(F2,X2)))),Ns,NsR),
    make_tree(NsR,T).
```

```

% insert(n(F,X),Ns,NsR) :- insert the node n(F,X) into Ns such that the
%   resulting list NsR is again sorted with respect to the frequency F.

insert(N,[],[N]) :- !.
insert(n(F,X),[n(F0,Y)|Ns],[n(F,X),n(F0,Y)|Ns]) :- F < F0, !.
insert(n(F,X),[n(F0,Y)|Ns],[n(F0,Y)|Ns1]) :- F >= F0, insert(n(F,X),Ns,Ns1).

% traverse_tree(T,Cs) :- traverse the tree T and construct the Huffman
%   code table Cs,

traverse_tree(T,Cs) :- traverse_tree(T,"",Cs1-[]), sort(Cs1,Cs).

traverse_tree(n(_,A),Code,[hc(A,Code)|Cs]-Cs) :- atom(A). % leaf node
traverse_tree(n(_,s(Left,Right)),Code,Cs1-Cs3) :- % internal node
    atom_concat(Code,'0',CodeLeft),
    atom_concat(Code,'1',CodeRight),
    traverse_tree(Left,CodeLeft,Cs1-Cs2),
    traverse_tree(Right,CodeRight,Cs2-Cs3).

% The following predicate gives some statistical information.

huffman(Fs) :- huffman(Fs,Hs) , nl, report(Hs,5), stats(Fs,Hs).

report([],_) :- !, nl, nl.
report(Hs,0) :- !, nl, report(Hs,5).
report([hc(S,C)|Hs],N) :- N > 0, N1 is N-1,
    writef('%w %8l ',[S,C]), report(Hs,N1).

stats(Fs,Cs) :- sort(Fs,FsS), sort(Cs,CsS), stats(FsS,CsS,0,0).

```

```

stats([],[],FreqCodeSum,FreqSum) :- Avg is FreqCodeSum/FreqSum,
    writef('Average code length (weighted) = %w\n',[Avg]).
stats([fr(S,F)|Fs],[hc(S,C)|Hs],FCS,FS) :-
    atom_chars(C,CharList), length(CharList,N),
    FCS1 is FCS + F*N, FS1 is FS + F,
    stats(Fs,Hs,FCS1,FS1).

```

### Output:

```

?- huffman([fr(a,45),fr(b,13),fr(c,12),fr(d,16),fr(e,9),fr(f,5)]).
a 0          b 101          c 100          d 111          e 1101
f 1100
Average code length (weighted) = 2.24
true

```

**Conclusion :** In this Experiment I used Prolog programming to solve the given problem .