INSTRUCTION MANUAL

EXPERT SYSTEM LAB

(CS-430)

Prepared by

Er. Supreet Kaur
Assistant Professor (CSE)

GURU NANAK DEV ENGINEERING COLLEGE
LUDHIANA – 141006

DECLARATION

This Manual of Expert System (CS-430) has been prepared by me as per syllabus of Expert System (CS-430).

Signature

SYLLABUS

Departmental Elective - III
CS-430 EXPERT SYSTEMS LAB

External Marks: 20 L T P
Internal Marks: 30 - - 2
Total Marks: 50

Students are required to develop expert system for various industries/real life problems.

- Medical Diagnosis
- Trouble Shooting of Computer Systems and PCs.
- Electrical Machines
- Chemical Processes
- Structure Analysis

LIST OF EXPERIMENTS

11	Write a program to solve tower of hanoi.	16
12	Develop an Expert System of Moderate Complexity.	

EXPERIMENT NO. 1

AIM:- Introduction to Prolog.

Prolog, which stands for PROgramming in LOGic, is the most widely available language in the logic programming paradigm. Logic and therefore Prolog is based the mathematical notions of relations and logical inference. Prolog is a declarative language meaning that rather than describing how to compute a solution, a program consists of a data base of facts and logical relationships (rules) which describe the relationships which hold for the given application. Rather then running a program to obtain a solution, the user asks a question. When asked a question, the run time system searches through the data base of facts and rules to determine (by logical deduction) the answer.

Among the features of Prolog are `logical variables' meaning that they behave like mathematical variables, a powerful pattern-matching facility (unification), a backtracking strategy to search for proofs, uniform data structures, and input and output are interchangeable.

Often there will be more than one way to deduce the answer or there will be more than one solution, in such cases the run time system may be asked find other solutions. backtracking to generate alternative solutions. Prolog is a weakly typed language with dynamic type checking and static scope rules.

Prolog is used in artificial intelligence applications such as natural language interfaces, automated reasoning systems and expert systems. Expert systems usually consist of a data base of facts and rules and an inference engine, the run time system of Prolog provides much of the services of an inference engine.

The Structure of Prolog Programs

- A Prolog program consists of a database of facts and rules, and queries (questions).
    - Fact: ... .
    - Rule: ... :- ... .
    - Query: ?- ... .
    - Variables: must begin with an upper case letter.
    - Constants: numbers, begin with lowercase letter, or enclosed in single quotes.

Lists: append, member

list([]).	
list([X|L])	:- [list(L).
Abbrev:	$[X_1\|[...[X_n\|[]...] =$ $[X_1,...X_n]$
append([],L,L).	
append([X|L1],L2,[X|L12])	:- append(L1,L2,L12).
member(X,L)	:- concat(_,[X|_],L).

Ancestor

```
ancestor(A,D) :- parent(A,B).
ancestor(A,D) :- parent(A,C),ancestor(C,D).
```

but not

```
ancestor(A,D) :- Ancestor(A,P), parent(P,D).
```

  - *since infinite recursion may result.*

- Depth-first search: Maze/Graph traversal
  A database of arcs (we will assume they are directed arcs) of the form:

```
a(node_i,node_j).
```

- Rules for searching the graph:

```
go(From,To,Trail).
go(From,To,Trail) :- a(From,In), not visited(In,Trail), go(In,To,[In|Trail]).
visited(A,T)      :- member(A,T).
```

- I/O: terms, characters, files, lexical analyzer/scanner
  - read(T), write(T), nl.
  - get0(N), put(N): ascii value of character
  - name(Name,Ascii_list).
  - see(F), seeing(F), seen, tell(F), telling(F), told.
- Natural language processing: Context-free grammars may be represented as Prolog rules. For example, the rule

```
sentence ::= noun_clause verb_clause
```

- *can be implemented in Prolog as*

```
sentence(S)  :- append(NC,VC,S), noun_clause(NC), verb_clause(VC).
```

or in DCG as:

```
Sentence     -> noun_clause, verb_clause.
?- sentence(S,[]).
```

- Note that two arguments appear in the query. Both are lists and the first is the sentence to be parsed, the second the remaining elements of the list which in this case is empty.

A Prolog program consists of a data base of facts and rules. There is no structure imposed on a Prolog program, there is no main procedure, and there is no nesting of definitions. All facts and rules are global in scope and the scope of a variable is the fact or rule in which it appears. The readability of a Prolog program is left up to the programmer.

Facts

This chapter describes the basic Prolog facts. They are the simplest form of Prolog predicates, and are similar to records in a relational database. As we will see in the next chapter they can be queried like database records.

The syntax for a fact is

```
        pred(arg1, arg2, ...  argN).
```

where

pred

    The name of the predicate

arg1, ...

    The arguments.

Viva questions:

1. What is Prolog?
2. What is the difference between clauses and predicates?
3. How can you write facts and rules?

Assignments:

1. What are the different data types in prolog
2. 2. Explain Data Structure in Prolog.

EXPERIMENT NO. 2

/*Write a program in Prolog that maintain knowledge base consisting of parent relationship, gender (male, female), and rules for offspring, mother, father, son and daughter.*/

```
domains
        person=symbol
predicates
        mother(person,person)
        father(person,person)
        parent(person,person)
        son(person,person)
        daughter(person,person)
        offspring(person,person)
        female(person)
        male(person)
clauses
        parent(jim,bob).
        parent(janet,jim).
        parent(eileen,gloria).
        parent(julie,martha).
        parent(rick,bob).
        female(martha).
        female(gloria).
        female(janet).
        female(eileen).
        female(julie).
        male(bob).
        male(jim).
        male(rick).
        offspring(X,Y):-parent(Y,X).
        mother(X,Y):-parent(X,Y),female(Y).
        father(X,Y):-parent(X,Y),male(Y).
        son(X,Y):-offspring(X,Y),male(Y).
        daughter(X,Y):-offspring(X,Y),female(Y).
```

OUTPUT:

```
Goal: daughter(eileen,gloria)
No
Goal: daughter(gloria,eileen)
Yes
Goal: son(bob,Son)
Son=jim
Son=rick
2 Solutions
Goal: offspring(martha,julie)
Yes
```

```
Goal: father(janet,jim)
Yes
Goal: mother(eileen,Mother)
Mother=gloria
1 Solution
```

EXPERIMENT NO. 3

/*Write a program in Prolog that maintains knowledge base about the employee name, birth, job, and marital status in form of structure.*/

```
domains
        person=name(first,mi,last)
        first,last=symbol
        mi=char
        position=job(title,department)
        title,department=symbol
        citizenship=birth(date,city)
        date,city=symbol
        education=grad(college,degree,year)
        college,degree=symbol
        year=integer
        marital_status=married(spouse,date); divorced(date); single()
        spouse=symbol
        salary=real
predicates
        employee(person,citizenship,education,position,marital_status,salary)
clauses
        employee(name(john,'s',baker),
                birth("01/03/1975",harrison),
                grad("Texas A&M","BA Economics",1990),
                job(manager,sales),
                divorced("06/07/1999"),
                37500).
        employee(name(julie,'h',clinton),
                birth("15,08,1981",canada),
                grad("calgary univ","BTech ECE",2004),
                job(engineer,development),
                married("Boby Clinton",canada),
                35000).
        employee(name(white,'e',patrick),
                birth("09/12/1958","San Francisco"),
                grad("Fresno state","BA Pylosophy",1981),
                job(manager,customer_support),
                single,
                32900).
```

OUTPUT:

```
Goal: write("Name=John\n"),employee(name(john,_,_),birth(_,City),_,job(Design
ation,_),Marital_status,_)
Name=John
City=harrison, Designation=manager, Marital_status=divorced("06/07/1999")
1 Solution Goal
```

EXPERIMENT NO. 4

```
/*Write a program in Prolog to find greatest of three numbers.*/
trace
domains
        num1,num2,num3,max=integer
predicates
        max_num(num1,num2,num3,max)
clauses
        max_num(X,Y,Z,T):-X>Y,X>Z,!,T=X.
        max_num(X,Y,Z,T):-Y>X,Y>Z,!,T=Y.
        max_num(X,Y,Z,T):-T=Z.
```

<u>OUTPUT:</u>

```
Goal: max_num(12,11,34,Greatest)
Greatest=34
1 Solution
Goal: max_num(56,11,11,Greatest)
Greatest=56
1 Solution
Goal: max_num(23,12,23,Greatest)
Greatest=23
1 Solution
Goal:
```

EXPERIMENT NO. 5

```
/*Write a program in Prolog to solve the equation X=(20*Y+5*Z)/4.*/

domains
	x,y,z=real
predicates
	solve(x,y,z)
clauses
	solve(X,Y,Z):-X=(20*Y+5*Z)/4.
```

OUTPUT:

```
Goal: solve(4,3,2)
No
Goal: solve(X,3,2)
X=17.5
1 Solution
Goal: solve(17.5,3,2)
Yes
Goal:
```

EXPERIMENT NO. 6

```
/*Write a program in Prolog that maintains the knowledge base about the students name,
registration number, course, grade in form of structures.*/

domains
        person=name(first,mi,last)
        first,last=symbol
        mi=char
        reg_no=integer
        course=symbol
        grade=char
predicates
        student(person,reg_no,course,grade)
clauses
        student(name("Rekha",'R',"Goyal"),
                121,"computer science",'A').
        student(name("Amanpreet",'K',"Gill"),
                122,"computer science",'A').
        student(name("Hansdeep",'K',"Hans"),
                123,"computer science",'A').
```

OUTPUT:

```
Goal: student(name(Name,_,_),Reg_no,Course,_)
Name=Rekha, Reg_no=121, Course=computer science
Name=Amanpreet, Reg_no=122, Course=computer science
Name=Hansdeep, Reg_no=123, Course=computer science
3 Solutions
Goal:
```

EXPERIMENT NO. 7

/*Write a program in Prolog that maintains the collection of facts related to people and their favorite outdoor activity {likes(X,Y)}, and the type of work they do for living {works(X,Y)}. Add appropriate rules to knowledge base to find whether a person likes the work he does or not.*/

```
domains
        name,hobby,job=symbol
predicates
        likes(name,hobby)
        works(name,job)
        like_work(name,job)
clauses
        likes(chris,swimming).
        likes(john,fishing).
        likes(leslie,rafting).
        likes(jackle,tennis).
        likes(susan,fishing).
        likes(peter,jogging).
        likes(ted,jogging).
        likes(anil,gardening).
        works(chris,programmer).
        works(john,mailing).
        works(leslie,artist).
        works(jackle,tennis).
        works(susan,secretary).
        works(peter,banker).
        works(ted,cook).
        works(anil,gardening).
        like_work(X,Y):-likes(X,Z),works(X,Y),Z=Y.
```

OUTPUT:

```
Goal: write("Jackle likes his work: "),like_work(jackle,tennis)
Jackle likes his work: Yes
Goal: write("Susan's work and liking is same: "),like_work(susan,secretary)
Susan's work and liking is same: No
Goal: likes(ted,Ted_likes)
Ted_likes=jogging
1 Solution
Goal:
```

EXPERIMENT NO. 8

```
/*Write a program to
        a. Count the length (no of elements) of array
        b. To find the sum of N numbers
        c. To find the average of N numbers */

domains
        len,sum=integer
        avg=real
        list=integer*
predicates
        length(list,len)
        addition(list,sum)
        average(list,avg)
clauses
        /*To find the length of the list*/
        length([],0).
        length([_|Tail],Length):-length(Tail,Length1),Length=Length1+1.

        /*To find the sum of the list*/
        addition([],0).
        addition([Head|Tail],Sum):-addition(Tail,Addtail),Sum=Head+Addtail.

        /*To find the average of a list*/
        average(List,Avg):-length(List,L),addition(List,S),Avg=S/L
```

OUTPUT:

```
Goal: length([3,2,5,4,6],Length_is)
Length_is=5
1 Solution
Goal: addition([5,2,7,3,4],Sum_is)
Sum_is=21
1 Solution
Goal: average([4,6,3,7,4],Average_is)
Average_is=4.8
1 Solution
Goal.
```

EXPERIMENT NO. 9

```
/*Write a program to implement following basic list functions:
        a. Appending an element to list
        b. Searching an element from the list
        c. Merge two lists.
        d. Deleting an element from a list
        e. Find all possible permutations among the elements of the list.*/
trace
domains
        object=symbol
        list=object*
predicates
        append(object,list,list)
        search(object,list)
        merge(list,list,list)
        delete(object,list,list)
        perm(list,list)
clauses
        /*Append an element in a list*/
        append(X,List,[X|List]).

        /*To search an element in a list*/
        search(X,[X|_]).
        search(X,[_|Tail]):-search(X,Tail).

        /*Merge two lists*/
        merge([],List,List).
        merge([Head|List1],List2,[Head|List3]):-merge(List1,List2,List3).

        /*Delete an element from a list*/
        delete(X,[X|List],List).
        delete(X,[Y|Tail],[Y|Tail1]):-delete(X,Tail,Tail1).

        /*Permutations for a list*/
        perm([],[]).
        perm(List,[Head|Tail]):-delete(Head,List,List1),perm(List1,Tail).
```

<u>OUTPUT:</u>

```
Goal: append(hello,[how,are,you],L)
L=["hello","how","are","you"]
1 Solution
Goal: search(are,[hello,how,are,you])
Yes
Goal: search(b,[a,c,k,j,l])
```

```
No
Goal: merge([a,b,c],[x,y,z],L)
L=["a","b","c","x","y","z"]
1 Solution
Goal: delete(y,[a,b,c,d,x,y,z],L)
L=["a","b","c","d","x","z"]
1 Solution
Goal: perm([c,a,t],L)
L=["c","a","t"]
L=["c","t","a"]
L=["a","c","t"]
L=["a","t","c"]
L=["t","c","a"]
L=["t","a","c"]
6 Solutions
Goal
```

EXPERIMENT NO. 10

/*Write a program to seperate all the character from the string.*/

```
domains
        list=char*
predicates
        string_to_list(list,string)
clauses
        string_to_list([],"").
        string_to_list([Headchar|Tail],String):-
                        frontchar(String,Headchar,Remstring),
                        string_to_list(Tail,Remstring).
```

OUTPUT:

```
Goal: string_to_list(List_is,"Hansdeep Kaur")
List_is=['H','a','n','s','d','e','e','p',' ','K','a','u','r']
1 Solution
Goal
```

EXPERIMENT NO. 11

```
/*Write a program to solve tower of hanoi.*/

domains
        num,peg=integer
predicates
        hanoi(num)
        move(num,peg,peg,peg)
        inform(peg,peg)
clauses
        hanoi(N):-move(N,3,1,2).
        move(0,_,_,_):-!.
        move(N,A,B,C):-M=N-1,move(M,A,C,B),inform(A,B),move(M,C,B,A).
        inform(X,Y):-write("move disc from",X,"to",Y,"\n").
```

OUTPUT:

```
Goal: hanoi(3)
move disc from3to1
move disc from3to2
move disc from1to2
move disc from3to1
move disc from2to3
move disc from2to1
move disc from3to1
Yes
```