Introduction to Artificial Intelligence

The Artificial Intelligence (AI) can be defined as the study of how to make computers do things which, at the moment, people do better. Or can be defined as the branch of computer science that is concerned with automation of intelligent behavior. Some of the task domains of Artificial Intelligence includes are listed below:

1. Formal Tasks such as:-
    - Games (Chess, Backgammon, Checkers – Go).
    - Mathematics (Geometry, Logic, Integral calculus).
2. Mundane Tasks such as :-
    - Perception (Vision, Speech). Perceptual tasks are difficult because they involve analog (rather than digital) signals; the signals are typically very noisy.
    - Natural Language (Understanding, Generation, Translation): - the problem of understanding spoken language is a perceptual problem and is hard to solve. It is extremely difficult because it required to know a lot about of the language (Vocabulary, Grammar).
    - Commonsense reasoning: - it is includes reasoning about physical objects and their relationships to each other. Also reasoning about actions and their consequences.
    - Robot control
3. Experts Tasks
    - Engineering (Design, fault Finding, Manufacturing planning).
    - Scientific analysis
    - Medical diagnosis
    - Financial analysis

AI Technique

- Search :- Providing a way of solving problems for which no more direct approach is a available, as well as a framework into which any direct techniques that are available can be embedded.
- Use of Knowledge :- provides a way of solving complex problems by exploiting the structures of objects that are involved.
- Abstraction :- Provides a way of separating important features and variations from the many unimportant ones that would otherwise overwhelm any process.

Introduction to Programming in Logic

There are two types of programming language paradigms:-

1) Procedural:- Traditional programming languages are said to be procedural. In procedural language the programmer has to specify in detail how to solve problem. FORTRAN, C, and even – object oriented languages fall under this general approach.
2) Declarative: in declarative paradigm, declarative programming language means that rather than describing how to compute the solution, a program consist of database of facts and logical relationships (rules), which describes the relationships. Which holds for given application. Prolog and LDL are examples of declarative languages. ***Logic programming is a declarative paradigm.***

The logic program dealing with relation rather than functions. Logic programming functionality can be represented as the following:-

Algorithm = Logic + Control

The Logic refers to the facts and rules specifying what the algorithm does, and the control refers to how algorithm can be implemented by applying the rules in a particular order.

The idea of logic programming is to use a computer for drawing conclusions from declarative descriptions.

Definitions of Prolog (**PRO**gramming in **LOG**ic)

Prolog: is a declarative programming language, it considered one of the most widely used programming languages in Artificial Intelligence (AI) research,

A prolog program consists of facts and rules. There is no structure imposed on a Prolog program, there is no main procedure, and there is no nesting

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 main features of Prolog are:

1. Rule based programming: the rule based programming allows the program code to be written in a form which is more declarative than procedural.
2. Built in a pattern matching: this is an important features of prolog.
3. Backtracking execution: Backtracking provides for the flow of control in the program.
4. Ability to deduction.

Applications of Prolog

The main applications of Prolog are:

- Intelligent database retrieval
- Natural language understanding
- Expert systems (ES)
- Specification language
- Machine learning
- Automated reasoning
- Problem solving

Prolog Program

Prolog program consists of the following parts :-

1. **Domains:** define global parameter used in the program. Like

   Domains

   I= integer

   C= char

   S = string

   R = real

2. **Predicates:** define rule and fact used in the program (declaration), like

   mark(symbol, integer).

male (string).

parent (string, string).

xor(integer, integer, integer).

3. **clauses:** define the body of the program

parent(jane, alan). ←——————Fact

mother(P1,P2):- parent(P1,P2), female(P1). ←——Rule

A clause consists of a *head* and sometimes a *body*. Facts don't have a body because they are always true. A predicate head consists of a *predicate name* and sometimes some *arguments* contained within brackets and separated by commas.

parent(jane, alan).

4. **Goal:** can be internal or external, internal goal written after clauses portion, external goal supported by the prolog compiler if the program syntax is correct. This portion contains the rule that drive the program execution.

Programming in Prolog

• Declaring some facts about object and their relationships.

• Defining some rules about objects and their relationships.

• Asking questions about objects and their relationships

So, prolog language can be considered as a store house of facts and rules, and it uses the facts and rules to answer questions.

The structure of Prolog language

A Prolog program consists of database of facts and rules, and queries(Question):-

1. Facts

Prolog consists of as series of facts and rules. Facts are either consist of a particular item, or relation between items. A fact in every language is often a

proposition like "Gold is valuable". Facts describe explicit relationships between objects and properties objects might have.

Syntax of fact:

1. The name of all relationship and objects must begin with a lower-case letter, for example likes (john, mary).
2. The relationship is written first, and the objects are written separated by commas, and enclosed by a pair of round brackets.
   Like (john, mary)
3. The full stop character '.' Must come at the end of fact.
4. Objects also begin with lowercase letters.

Examples:

Gold is valuable	valuable (gold).
Jane is female	female (jane).
John owns gold	owns (johns, gold).
Johns is the father of Mary	father (john, marry).

The names of objects that are enclosed within the round brackets are called arguments. And the name of relationship called predicates.

Relationship has arbitrary number of argument. If we want to define predicate called play, were we mention two players and a game they play with each other, it can be:

play (john, Mary, football).

2. Rules

Rule consists of a head (a predicate) and a body (a sequence of predicates separated by commas). To build a rule, head can be represented as conclusion and body can be represented as condition.

The word ***if*** used after the head and represented as "**:-**" which separates the Head and body, like every Prolog expression, a rule has to be terminated by a dot.

The syntax of ***if*** statement

If (condition) then (conclusion)

[conclusion :- condition]. Rule

For Example:

It will rain if the sky is cloudy

conclusion condition

represent both as fact like:

wheatear(rain).

cloudy(sky).

wheatear(rain) :- cloudy(sky). Rule

****

I use the umbrella if there is rain

Conclusion condition

Represent both as fact like:

wheatear (rain).

use (umbrella)

use (Iam, umberella):-whether (rain).

3. Queries (Questions)

A query in prolog is the action of asking the program about information contain within its database(facts and rules).

Type of questing in the goal

There are three type of question in the goal summarized as follow:

1. Asking with constant: prolog matching and return Yes/No answer.

male(ali).

male(ahmed).

male(khalid).

male(samir).

Goal:

male(ali). Ans:- Yes

male(suha) . Ans :- No.

2. Asking with constant and variable: prolog matching and produce result for the variable.

parent(ali, ahmed).

parent (ali, suha).

parent(ali, samir).

parent(ahmed, khalid).

Goal:

parent(X, ahmed). Ans:- X= ali.

parent(ali,X). Ans:- X= ahmed, X= suha, X= samir.

3. Asking with variable: prolog produce all possible results.

parent(ali, ahmed).

parent (ali, suha).

parent(ali, samir).

parent(ahmed, khalid).

Goal:

parent(X,Y). Ans:- X=ali, Y= ahmed.

X=ali, Y= suha.

X=ali, Y=samir.

X=ahmed, Y= khalid.

Variables in Prolog

If we want to get more interest information about fact or rule, we can use variable to get more than Yes/No answer.

1. variables dose not name a particular object but stand for object that we cannot name.
2. variable name must begin with ***capital letter***.
3. using variable we can get all possible answer about a particular fact or rule.
4. variable can be either bound or not bound.

Variable is bound when there is an object that the variable stands for.
The variable is not bound when what the variable stand for is not yet known.

Data Types in Prolog.

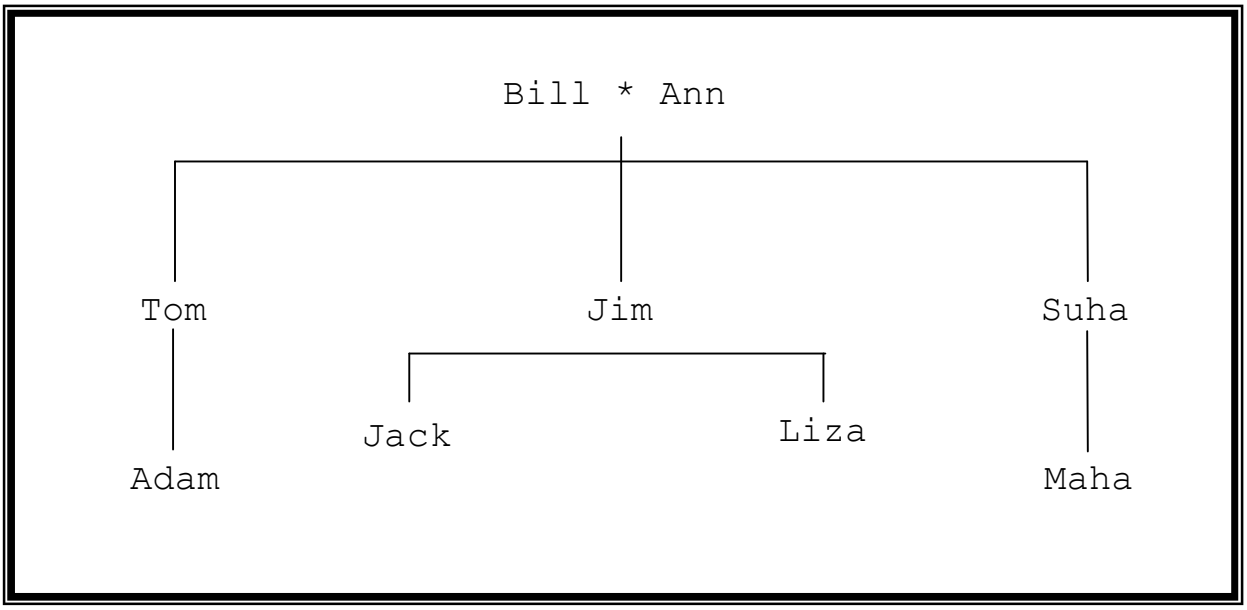
Prolog supports the following data type to define program entries.

1. Integer: to define integer value like 1, 20, 0,-3,-50, etc.
2. Real: to define the decimal value like 2.4, 3.0, 5,-2.67, etc.
3. char: to define single character, the character can be of type small letter or capital letter or even of type integer under one condition it must be surrounded by single quota. For example, 'a','C','1'.
4. string: to define a sequence of character.
5. Symbol: is similar to string, it deals with sequences of character, or single character.

For example:

```
domains
I=integer.
S=string.
Y=char.
```

Example computes the relation of this family




```
domains
  X=string.
predicates
  male(X).
  female(X).
  parent(X,X).
  father(X,X).
  mother(X,X).
  son(X,X).
  daughter(X,X).
  brother(X,X).
  sister(X,X).
  grandfather(X,X).
  grandmother(X,X).
  grandchild(X,X).
  uncle(X,X).
  aunt(X,X).
  cousin(X,X).
```

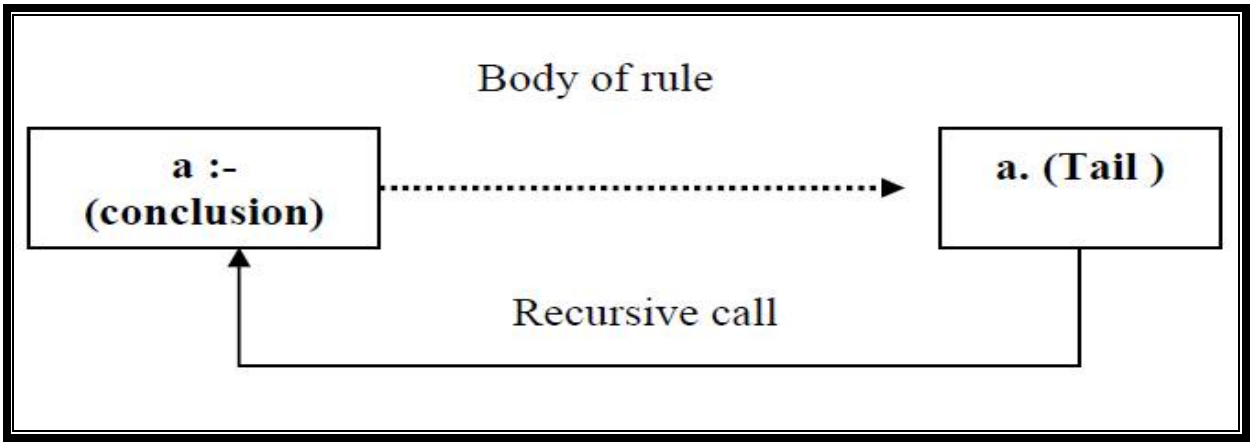
```
clauses
  male(bill).
  male(tom).
  male(jim).
  male(jack).
  male(adam).
  female(ann).
  female(suha).
  female(liza).
  female(maha).
  parent(bill,tom).
  parent(bill,jim).
  parent(bill,suha).
  parent(ann,tom).
  parent(ann,jim).
  parent(ann,suha).
  parent(jim,jack).
  parent(jim,liza).
  parent(suha,maha).
  parent(tom,adam).
  father(X,Y):- parent(X,Y),male(X).
  mother(X,Y):- parent(X,Y),female(X).
  son(X,Y):- parent(Y,X), male(X).
  daughter(X,Y):-parent(Y,X), female(X).
  brother(X,Y):- father(Z,X),father(Z,Y),X<>Y,male(X).
  sister(X,Y):- father(Z,X),father(Z,Y),X<>Y,
  female(X).
  grandfather(X,Y):-parent(X,Z),parent(Z,Y),male(X).
  grandmother(X,Y):-parent(X,Z),parent(Z,Y),female(X).
```

```
grandchild(X,Y):-?.
uncle(X,Y):-?.
aunt(X,Y):-?.
cousin(X,Y):-?.
```

Recursion in Prolog Language:-

In prolog language there are no iterative constructs (for, while, …,etc), instead of which it uses ***Recursion***. The recursion in any language is a function that can call itself until the goal has been succeed. In Prolog, recursion appears when a predicate contain a goal that refers to itself. There are two types of recursion:

1.Tail Recursion: We place the predicate that cause the recursion in the tail of the rule as shown below:



****

Example 1: Write a prolog program to print the number from n to 1.

```
domains
 I=integer.
predicates
 print(I).
clauses
 print(0):-!.
 print(N):- write(N),nl,N1=N-1, print(N1).

Goal: print(10).
10
```

```
9
8
7
6
5
4
3
2
1
yes
```

****

Homework: Write a prolog program to print the number from 1 to n.

****

Example 2: Write a prolog program to find the factorial of 5! = 5*4*3*2*1

```
domains
I= integer
predicates
fact( I, I, I)
clauses
fact(1, F, F):-!.
fact(N,F,R):- F1=F*N , N1=N-1, fact(N1,F1,R).

Goal: fact(5,1,F).
Output: F = 120
```

****

homework: Write a prolog program to find the s = n+(n-1)+(n-2)+…+1.

****

2. Non Tail Recursion (Stack Recursion)

is recursion in which the recursive call is not the last step in the procedure. This type of recursion use the stack to hold the value of the variables till the recursion is complete. The statement is self – repeated as many times as the number of items in the stack. This type of recursion using less number of variable than Tail recursion.

****

Example 4: factorial program using non-tail recursion.

```
Predicates
Fact ( integer, integer)
Clauses
Fact (0,1) :-!.
Fact (N,F):- N1= N-1, fact (N1,F1), F= N1*F1.

Goal:
Fact (5,F)
Output:
F =120.
```

****

Example 5: power program using non-tail recursion.

```
Predicates
Power (integer, integer, integer)
Clauses
Power (_,0,1):-!.
Power (X,Y,Z) :- Y1=Y -1, power (X,Y1,Z1), Z= X*Z1.

Goal:
Power (3,2,Z)
```

```
Output
Z = 9.
```

****

Homework

1) By using the non tail recursion print the number from 1 to n.

2) By using the non tail recursion print the number from n to 1.

3) By using the non tail recursion compute the following

- S= 1/(N)!+1/(N-1)!+…..+1/(1)!.
- S=1- (2^n/2!)+…..±(X^n/X!).

Built in Function in prolog

Prolog has a rang of built in function, the table below explain some of these function and predicates.

Function	Description
X mod Y	This function return the remainder of x divided by Y.
X div Y	This function return the value of division X by Y.
abs(X)	return the absolute value of X
cos(x)	return the cosine value of angle X in rad.
sin(x)	return the sin value of X
tan(X)	return the tan value of X
arctan(X)	return the arc tan of X
exp(X)	return the exponential of x
log(X)	logarithm of X base 10
sqrt(X)	square root of X
ln(X)	logarithm of X with base e

Read and write function

Read function:

`readint(Var)` : read integer variable.

`readchar(Var)` : read character variable.

`readreal(Var)` : read real(decimal) variable.

`readln(Var)` : read string.

Write function

`Write(Var)` : write variable of any type.

****

Example 1: write prolog program to read integer value and print it.

```
Domains
   I = integer
Predicates
   print.
Clauses
   Print:- write ("please read integer number"),
    readint(X), write("you read",X).

Goal
  Print.
  Output:
  Please read integer number 4
  You read 4
```

Cut Function

Sometimes it is desirable to selectively turn off backtracking. Prolog provides a predicate that performs this function. It is called the cut, represented by an ***exclamation point (!)*** .

The cut effectively tells Prolog to freeze all the decisions made so far in this predicate. That is, if required to backtrack, it will automatically fail without trying other alternatives.

```
Domains
  I= integer.
Predicates
```

```
    no( I )
Clauses
    no(5):-!.
    no(7).
    no(10).

Goal: no (X).
Output: X=5.
```

```
domains
  S=string.
predicates
  parent(X,X).
  male(X).
  father(X,X).
clauses
  male(ali).
  male(ahmed).
  male(khalid).
  parent(ali, ahmed).
  parent(ali, khalid).
  father(X,Y):-parent(X,Y), male(X),!.

Goal: father(X,Y).
Output: X= ali, ahmed.
```

```
1 solution
```

Fail Structure in Prolog.

The fail predicates causes the failure of the rule and this will be forever, nothing can change the statement of this predicate. The **fail** function is used to enforce backtracking, place always in the end of rule, produce false and can be used with internal goal to produce all possible solution.

Ex.

```
domains
  Y=integer.
predicates
  x(Y).
clauses
  x(1).
  x(2).
  x(3).
  loop:-x(A), write(A),fail.

Goal:-loop
Output:1 2 3 No
```

String in Prolog.

Prolog provides several standard predicates for powerful and efficient string manipulations. This section summarizes the standard predicates available for string manipulating and type conversion.

1. **str_len** (String ,Length) **:** Determines the length of String.

   ```
   str_len("prolog",X)
   X=6.
   ```

2. **str_int** (String ,Integer) **:** Converts a string of one character to ASCII code or vice versa.

   ```
   str_int("A",X)
   X=65.
   ```

3. **char_int** (char,integer)**:** Converts a character to ASCII code or vice versa.

   ```
   char_int('A',X)
   X=65.
   ```

4. **str_char(string,char):** convert the string (of one char) to char and the opposite.

   ```
   str_char("A",X)
   X='A'
   str_char(X,'A')
   X="A"
   ```

5. **isname (string):** test if the content of the string is name or not

   ```
   isname("s2") return YES.
   isname("4r") return NO.
   ```

6. **frontchar(String,Char,RestString) (string,char,string):** Extracts the first character from a string, the remainder is matched with Rest String.

   ```
   frontchar ("prolog", C, R).
   ```

```
C='p', R="rolog".
```

7. **fronttoken (String,Token,RestString) (string,string,string):** Skips all white space characters (blanks, tabs) and separates from the resulting string the first valid token. The remainder is matched with Rest String.

```
fronttoken ("complete prolog program",T,R)
T="complete", R="prolog program".
```

8. **frontstr (StrLen,String,FrontStr,RestStr):** Extracts the first **n** characters from a string.

```
frontstr(3,"cdab2000",T,R)
T="cda", R="b2000".
```

9. **concat (Str1,Str2,ResStr) (string,string,string):** Concat two string together to produce one string.

```
concat ("prolog","2011",R)
R="prolog2011".
```

10. **Upper_lower (string,string):** Convert the string in upper case (in capital letter) to the lower case (small letter) and the opposite.

```
Upper_lower(capital_letter,small_letter)
Upper_lower("ABC",X)
X="abc"
Upper_lower("Abc",X)
X="abc"
Upper_lower(X,"abc")
X="ABC"
```

Turbo Prolog Language Components.

Standard Names:- such as abs, log, sin, cos,…,etc.

Reserved words:- such as predicate, goal, fail, …,etc.

Identifiers:- all facts name and rules, as well as, the variables declared in domains field.

Variables:- such as X, Y, Age, …,etc.

String Constants:- such as "PROLOG means PROgramming in LOGic", "a12", 'y', …,etc.

Numeric Constants:- such as 60,-50, 3.14, …,etc.

Comments:- such as /* this is a comment*/, % this is a comment, …,etc.

List in Prolog

In prolog, a list is an data structure (object) that contains an arbitrary number of terms (any data types) within it and that can have any length, it is like array in another language. A list is either empty or it is a structure that has two components: the ***head H*** and ***tail T***.

Syntax of List

List always defined in the domains section of the program as follow:

```
domains
list = integer*
```

1) '*' refer to list object which can be of length zero or undefined.

2) The type of element list can be of any standard defined data type like integer, char … etc.

3) List element surrounded with square brackets and separated by comma as follow: L = [1, 2, 3, 4].

4) List consist of two parts head and tail, the **head represent the first element in the list** and the **tail represent the remainder** (i.e. head is an element but tail is a list). for the following list :

L = [1,2,3]

H = 1 T =[2,3]

H =2 T =[3]

H =3 T=[ ]

[ ] refer to empty list.

List can be written as [H|T] in the program

Example1 program to print all list element

```
domains
  L=integer*.
predicates
  write_list(L).
clauses
  write_list([]):-!.
  write_list([H|T]):-write(H),nl,write_list(T).
```

****

Example2 program to print Head and Tail of any List

```
domains
  L=integer*.
predicates
  write_list(L).
clauses
  write_list([]):-!.
```

```
  write_list([H|T]):-write("The Head=",H),nl,
write("The Tail =",T),nl,write_list(T).
```

****

Example3 program to compute the length of any list

```
domains
  L=integer*.
predicates
  len(L,Integer).
clauses
  len([],0):-!.
  len([_|T],L):-len(T,L1), L=L1+1.
```

****

Example 4 program to compute the maximum number in any list

```
domains
  L=integer*.
predicates
  max(Integer,L).
clauses
  max(X,[X]):-!.
  max(X,[H1,H2|T]):-H1>H2, max(X,[H1|T]).
  max(X,[H1,H2|T]):- H2>H1,max(X,[H2|T]).
```

****

H.W program to compute the minimum number in any list

****

Example5 program to delete the first element from any list

```
domains
  L=integer*.
predicates
  delete_first(L,L).
clauses
  delete_first([_|T],T):-!.
```

****

H.W. program to delete the last element of any list.

****

Example6 program to add new element to beginning of any list

```
domains
  L=integer*.
predicates
  add_first(integer,L,L).
clauses
  add_first(X,L,[X|L]):-!.
```

****

H.W program to add new element to the last of any list

****

Example7 program to delete specific element from any list // the list contain individual elements.

```
domains
  L=integer*.
predicates
  delete(integer,L,L).
clauses
```

```
delete(_,[],[]):-!.
delete(X,[X|T],T):-!.
delete(X,[H|T1],[H|T2]):-delete(X,T1,T2).
```

****

Example8 program to find specific element in a list

```
domains
  L=integer*.
predicates
  find(integer,L).
clauses
  find(_,[]):-write("The Element is Not Found"),nl,!.
  find(X,[X|_]):-write("The Element is Found"),nl,!.
  find(X,[_|T1]):-find(X,T1).
```

****

Example9 program to append two lists into one list

```
domains
  L=integer*.
predicates
  append(L,L,L).
clauses
  append([],[],[]):-!.
  append([],[H|T1],[H|T2]):- append([],T1,T2).
  append([H|T1],T2,[H|T3]):- append(T1,T2,T3).
```

****

H.W. program to append three lists in one list

****

Example10:-program to append two lists into one list without repetition the numbers.

```
domains
  L=integer*.
  X=integer.
predicates
  member(X,L).
  union(L,L,L).
clauses
  member(X,[X|_]):-!.
  member(X,[_|T]):-member(X,T).

  union([],T2,T2):-!.
  union([H|T1],T2,T3):-member(H,T2), union(T1,T2,T3).
  union([H|T1],T2,[H|T3]):-
not(member(H,T2)),union(T1,T2,T3).
```

****

Example10 program to divide a list of integer into two lists, the first one contains the even number, the second one contains the odd number.

```
domains
  L=integer*.
predicates
  divide(L,L,L).
clauses
  divide([],[],[]):-!.
  divide([H|T1],[H|T2],T3):-H mod  2 =0,
divide(T1,T2,T3).
```

```
  divide([H|T1],T2,[H|T3]):-H mod  2 =1,
divide(T1,T2,T3).
```

****

Example11 program to reverse a list.

```
domains
  L=integer*.
  X=integer.
predicates
  del_last(X,L,L).
  reverse(L,L).
clauses
  del_last(X,[X],[]):-!.
  del_last(X,[H|T1],[H|T2]):- del_last(X,T1,T2).

  reverse([],[]):-!.
  reverse(L1,[X|L2]):-
del_last(X,L1,L3),reverse(L3,L2).
```

****

Example12 program to sort a list descending

```
domains
  L=integer*.
  X=integer.
predicates
  max(X,L).
  delete(X,L,L).
  sort(L,L).
clauses
  max(X,[X]):-!.
```

```
  max(X,[H1,H2|T1]):- H1>H2, max(X,[H1|T1]).
  max(X,[H1,H2|T1]):- H2>H1, max(X,[H2|T1]).

  delete(X,[X|T1],T1):-!.
  delete(X,[H|T1],[H|T2]):-delete(X,T1,T2).

  sort([],[]):-!.
  sort(L1,[X|L2]):-
max(X,L1),delete(X,L1,L3),sort(L3,L2).
```

****

H.W. program to convert a list of char into a list of ASCII.

****

Files in Prolog

To deals with files in prolog language, you must do the following:-

1. define the file in define the file in the domains part.

```
domains
        file= first;
        file =second;
```

Symbolic File Name, used to deal with file in the program.

for every file there is two name, one used to deal with it in the program, while the other it is the real name for the file (represent the name that found on the hard disk.

2. open the file for different operation,

- open file for writing in it.

```
openwrite(Symbolic File Name, Real File Name)
```

this instruction cause to create new file when the file is not created before. Or open exist file with delete all it component. To write in file that is aleardy opened for writing we use the following instruction:-

writedevice (Symbolic File Name)

****

example of how to open file to store

```
domains
  file = first.
predicates
  goal1.
  goal2.
clauses
  goal1:- openwrite(first,"E:\\profile.txt"),goal2.
  goal2:- writedevice(first), write("Name
:\t"),readln(Name), Name <> "%",!,write(Name),nl,
        write("Age :\t"), readint(Age),
write(Age),nl, write("Tel :\t"), readint(Tel),
         write(Tel),nl,nl,goal2.
  goal2:- closefile(first).
```

****

This function used to close file, is necessary to write it.

- Open file for reading from it.

  To read from exist file we use the following instructions

openread(Symbolic File Name, Real File Name)

readdevice (Symbolic File Name)

example program to read from file

```
domains
  file = first.
predicates
  goal1.
  goal2.
clauses
  goal1:- openread(first,"E:\\profile.txt"),
          goal2.
  goal2:- readdevice(first), not(eof(first)),
readln(A), write(A),nl,goal2.
  goal2:- readdevice(keyboard),readchar(_).
```

****

- Open file for append text to it

To append text to exist file we use the following instruction

openappend(Symbolic File Name, Real File Name)

writedevice (Symbolic File Name)

****

example program to append text to file.

```
domains
  file = first.
predicates
  goal1.
  goal2.
clauses
  goal1:-
openappend(first,"E:\\profile.txt"),goal2.
```

```
goal2:-     writedevice(first),     write("Name
:\t"),readln(Name), Name <> "%",!,write(Name),nl,
          write("Age     :\t"),     readint(Age),
write(Age),nl, write("Tel :\t"), readint(Tel),
          write(Tel),nl,nl,goal2.
goal2:- closefile(first).
```

****

- Mode of File :- to know the mode of file we use the following instruction

filemode(Symbolic File Name, fm)

fm = 0 to text file

fm = 1 to binary file

****

- **ExistFile** this function check if the file is exist (created before) or not.

existfile("E:\f1.txt")

if exist it returns Yes, otherwise return No

****

- **renamefile** this function used to rename the file, it's format :-

renamefile(oldrealname, newrealname).

****

- **deletefile** this function used to delete the file, it's format :-

deletefile(RealFileName).

Database in Prolog Language

To deal with dynamic database in prolog we must declare data base in the database field in program

```
domains
   X=...
database
   no(X).
predicates
.
.
.
```

To add value to database we use *assert* instruction

To save database to file we use *save* instruction

To load database file for reading we use *consult* instruction

****

Example program to read number from keyboard and store in database file

```
domains
  X=integer.
database
  no(X).
predicates
  r
clauses
  r:- readint(X), X<>0, assert(no(X)),r.
  r:- save("input.db").
```

****

program to open database file and print its content

```
domains
  X=integer.
database
  no(X).
predicates
  r1
  r2
clauses
  r1:- consult("input.db"),r2.
  r2:- no(X), write(X),nl,fail.
  r2:-!.
```