1^{st} Class

2016-2017

Prolog Language

لغة البرولوك

أستاذة المادة : أ.م.د. إسراء عبد الأمير

Prolog Language

Lecture one:

Contents:

1. Introduction to prolog language

Prolog: is a computer programming language that is used for solving problems involves objects and relationships between objects.

Example:

"John owns the book"

Owns (john,book)   relationship(object1,object2)

The relationship has a specific order, johns own the book, but the book dose not owns john, and this relationship and its representation above called fact.

♦we are using rule to describe relationship between objects.

Example: the rule" two people are sisters if they are both female and have the same parents"

1. Tell us something about what it means to be sisters.

2. Tell us how to find if two people are sisters, simply: check to see if they are both female have the same parents.

Component of computer programming in prolog

Computer programming in prolog consist of:

1. Declaring some facts about object and their relationships.
2. Defining some rules about objects and their relationships.
3. Asking questions about objects and their relationships.

if we write our rule about sisters, we could then ask the questions whether Mary and Jane are sisters.

Prolog would search through what we told it about Mary and Jane, and come back with the answer Yes or No, depending on what we told it earlier.

So, we can consider prolog as a store house of facts and rules, and it uses the facts and rules to answer questions.

♦ prolog is a conversational language. Which means you and the computer carry out a kind of conversation, typing a letter from keyboard and displaying it at the screen, prolog work like this manner, prolog will wait for you to type in facts and rules that certain to the problem you want to solve? Then if you ask the right kind of questions prolog will work out the answers and show them.

2. Some of prolog language characteristics:

1. We can solve a particular problem using prolog in less no of line of code.
2. It’s an important tool to develop AI application and ES.
3. Prolog program consist of fact and rule to solve the problem and the output is all possible answer to the problem.
4. Prolog language is a descriptive language use the inference depend on fact and rule we submit to get all possible answer while in other language the

programmer must tell the computer on how to reach the solution by gives the instruction step by step.

3. Prolog language uses:

1. Construct NLI (Natural Language Interface).
2. Translate language.
3. Constructor symbolic manipulation language packages.
4. Implement powerfully database application.
5. Construct expert system programs.

4. Prolog language component

4.1 Facts

Is the mechanism for representing knowledge in the program.

<u>Syntax of fact:</u>

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.

Example:

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).

In prolog the collection of facts is called database.

4.2 Rules

Rules are used when you want to say that a fact depends on a group of other facts, and we use the following syntax:

1. One fact represents the head (conclusion).
2. The word if used after the head and represented as ":-'.
3. One or more fact represents the requirement (condition).

The syntax of if statement

If (condition) then (conclusion)

[Conclusion: - condition] rule

Example:

I use the umbrella if there is rain

Conclusion condition

Represent both as fact like:

Wheatear (rain).

Use (umbrella)

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

4.3 Questions

Question used to ask about facts and rules.

Question look like the fact and written under the goal program section while fact and rule written under clauses section.

Example: for the following fact owns (mary, book).

We can ask: Does mary own the book in the following manner:

Goal:

Owns (mary,book)

When Q is asked in prolog, it will search through the database you typed before, it look for facts that match the fact in the question.

Two fact matches if their predicates are the same and their corresponding argument are the same, if prolog finds a fact that matches the question, prolog will respond with Yes, otherwise the answer is No.

5. Variables

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

*variables dose not name a particular object but stand for object that we cannot name.

*variable name must begin with capital letter.

*using variable we can get all possible answer about a particular fact or rule.

*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.

Example:

Fact

*L*ike (john, mary).

Like (john, flower).

Like (ali, mary).

Questions:

1. Like (john,X)

X= mary

X = flower

2. like(X, mary)

X=john

3. Like(X, Y)

X=john Y=flower

X=john Y=mary

X=ali Y=mary

5. 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.

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

3. Asking with variable: prolog produce result.

Example:

Age(a,10).

Age(b,20).

Age(c,30).

Goal:

1.Age(a,X). ans:X=10 Type2

2.age(X,20). Ans:X=b Type2

3.age(X,Y). ans: X=a Y=10, X=b Y=20, X=c Y=30. Type3

4.Age(_,X). ans:X=10 , X=20, X=30. '_' means don't care Type3

5.Age(_,_). Ans:Yes Type1

Lecture Two:

1. Data type.
2. Program structure.
3. Read and write functions.
4. Arithmetic and logical operation.

1. data type

Prolog supports the following data type to define program entries.

1. **Integer:** to define numerical value like 1, 20, 0,-3,-50, ect.
2. **Real:** to define the decimal value like 2.4, 3.0, 5,-2.67, ect.
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’,’123’.
4. **string :** to define a sequence of character like “good” i.e define word or statement entries the string must be surrounded by double quota for example “computer”, “134”, “a”. The string can be of any length and type.
5. **Symbol:** anther type of data type to define single character or sequence of character but it must begin with small letter and don’t surround with single quota or double quota.

2. program structure

Prolog program structure consists of five segments, not all of them must appear in each program. The following segment must be included in each program predicates, clauses, and goal.

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

Domains

I= integer
C= char
S = string
R = real

2. **Data base:** define internal data base generated by the program
Database
Greater (integer)

3. **Predicates:** define rule and fact used in the program.
Predicates
Mark(symbol,integer).

4. **Clauses:** define the body of the program.. For the above predicates the clauses portion may contain Mark (a, 20).

5**.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.

2. mathematical and logical operation

a .mathematical operation:

operation	**symbol**
addition	+
subtraction	-
multiplication	*
Integer part of division	div
Remainder of division	mod

B .logical operation

operation	symbol
greater	>
Less than	<
Equal	=
Not equal	<>
Greater or equal	>=
Less than or equal	<=

3. Other mathematical function

Function name	**operation**
Cos(X)	**Return the cosine of its argument**
Sine(X)	**Return the sine of its argument**
Tan(X)	**Return the tranget of its argument**
Exp(X)	**Return e raised to the value to which X is bound**
Ln(X)	**Return the natural logarithm of X (base e)**
Log(X)	**Return the base 10 logarithm of log 10^x**
Sqrt(X)	**Return the positive square of X**
Round(X)	**Return the rounded value of X. Rounds X up or down to the nearest integer**
Trunc(X)	**Truncates X to the right of the decimal point**
Abs(X)	**Return the absolute value of X**

4. Read and write function

Read function:

readint(Var) : read integer variable.
Readchar(Var) : read character variable.
Readreal(Var) : read read (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
```

Example2: write prolog program that take two integer input us integer and print the greater.

```
Domains
     I = integer
Predicates
    Greater ( i,i)
Clauses
    Greater(X,Y):-  X>Y,write(“the greater is”,X).
    Greater(X,Y):-  write (“ the greater is “,Y).
Goal
    Greater(4,3).
```

Output:

```
    The greater is 4
```

H.W:

1. write prolog program that read any phrase then print it.

2.write prolog program that read an integer number then print it after multiplying it by any other integer like 5.

Lecture Three: More examples

This lecture present several example that intended to display various way to write prolog program, how to write if –else program ,divide problem into several parts then combine them in a single rule and how to write program describe specific problem.

Example 1: write prolog program to check if the given number is positive or negative.

Basic rule to check the number

```
If    X>=0    then
                  X is positive
           Else
                 X is negative
```

```
Domains
     I= integer
Predicates
     Pos_neg(i)
Clauses
     Pos_neg(X):-X>=0, write(“positive number”),nl.
     Pos_neg(_ ):-write(“negative number”),nl.

Goal
      Pos_neg(4)
```

Output:

```
     Positive number
```

Note: nl mean new line.

Example 2: write prolog program to check if a given number is odd or even.

Basic rule to check number

```
If X mod 2=0 then
        X is even number
   Else
       X is odd number
```

```
Predicates
     Odd_even(integer)
Clauses
      Odd_even(X):-X mod 2= 0, write (“even number”), NL.
      Odd_even(X):- write (“odd number”), nl.

Goal
     Odd_even(5)

Output
     Odd number
```

Example 3: write prolog program to combine both rule in example 1 and example2.

```
Domains
      I= integer
Predicates
      Pos_neg(i)
      Odd_even(i)
      Oe_pn(i)
Clauses
      Oe_pn(X):-pos_neg(X),odd_even(X).
      Odd_even(X):-X mod 2= 0, write(“ even number”),nl.
      Odd_even(X):- write(“odd number”),nl.
       Pos_neg(X):-X>=0, write(“positive number”),nl.
      Pos_neg(_ ):-write(“negative  number”),nl.


Goal
    Oe_pn(3)

Output:
     Odd number
     Positive number
```

Note: the rule of same type must be gathering with each other.

Example 4 : write prolog program to describe the behavior of the logical And gate.

Truth table of And gate

X	Y	Z
0	0	0
1	0	0
0	1	0
1	1	1

Sol 1:

```
Domains
     I= integer
Predicates
   And1(I, I , I )

Clauses
   And1(0,0,0).
   And1(0,1,0).
   And1(1,0,0).
   And1(1,1,1).

Goal
   And1 (0,1,Z)

Output:
    Z =0
```

Sol 2:

From the truth table we can infer the following rule:

```
     If  X= Y then
              Z= X
         Else
              Z =0
```

```
Domains
    I= integer
Predicates
    And1 (I ,I, I )
Clauses
     And1 (X,Y,Z):-  X=Y, Z=X.
     And1(X,Y,Z):-  X<> Y, Z=0.

Goal
   And1(0,0,Z)

Output
   Z=0
```

H.W

1. Write prolog program that read character and check if it’s a capital letter, small letter, digit or special character.
2. Modify prolog program in example 3 such that the value of X is read inside the program.
3. Write prolog program that describe the operation of logical Or gate.

Lecture four:

1. ***Cut and fail function***
2. ***Negation***

1. cut

Represented as "**!**" is a built in function always True , used to stop backtracking and can be placed any where in the rule, we list the cases where "!" can be inserted in the rule:

1 .R:-f1, f2,!.   "f1, f2 will be deterministic to one solution.
2. R:-f1,!,f2.   " f1 will be deterministic to one solution while f2 to all .
3. R:- !,f1,f2.   "R will be deterministic to one solution.

Example1 : program with out use cut.

```
Domains
    I= integer
Predicates
   No( I )
Clauses
  No (5).
  No (7).
  No (10).

Goal
   No (X).
```

Output:

```
     X=5
     X=7
     X=10
```

Example 2: program using cut.

```
Domains
    I= integer
Predicates
    No( I )
Clauses
    No (5):-!.
     No (7).
    No (10).
Goal
   No (X).

 Output:
      X=5.
```

Example3: program with out using cut.

```
Domains
     I =integer
     S = symbol
Predicates
    a (I )
    b (s )
    c ( I, s )
Clauses
    a(10).
    a(20)
    b(a)
    b(c)
     c (X, Y):- a (X), b (Y).

Goal
   c(X,Y).

Output:
     X= 10    Y=a
     X=10     Y=c

     X=20      Y=a
     X=20      Y=c
```

Example 4: using cut in the end of the rule.

```
Domains
     I =integer
     S = symbol
Predicates
    a(I )
    b (s )
   c (I, s )
Clauses
    a(10).
     a(20)
     b(a)
     b(c)
      c (X, Y):- a (X), b (Y),!.

Goal
    c(X,Y).

Output:
    X= 10    Y=a
```

Example 5: using cut in the middle of the rule.

```
Domains
     I =integer
     S = symbol
Predicates
    a(I )
    b (s )
    c ( I, s )
Clauses
     a(10).
     a(20)
     b(a)
     b(c)
     c (X, Y):- a (X),!, b (Y).

Goal
     c(X,Y).

Output:
     X= 10    Y=a
              Y=c
```

2. Fail

Built in function written as word “fail” 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.

Example 6:

```
Predicates
     Student ( symbol , integer)
      Printout.
Clauses
    Student (aymen,95).
    Student(zainab,44).
    Student(ahmed,60).

    Printout:-student(N,M),write(N,”    “,M),nl,fail.

Goal
   Printout.
```

Output:

```
      aymen   95
      zainab    44
      ahmed    60
        No
```

Example 7:

```
Predicates
     Student ( symbol , integer)
      Printout.
Clauses
    Student (aymen,95).
     Student(zainab,44).
     Student(ahmed,60).
```

```
  Printout:-student(N,M),write(N,”    “,M),nl,fail.
   Printout.
```

Goal

```
   Printout.
```

Output:

```
    aymen   95
    zainab    44
    ahmed    60
     Yes
```

H.w:

1. Trace the following clauses and find the output:

a. clauses

```
reading:- readchar(Ch),writ(Ch),Ch=’#’.
Reading.
```

b.clauses

```
Go.
Go:-go.
Reading:- go,readchar(Ch),write(Ch),Ch=’#,!.
```

1. *Use negation to define the different relation: diff(X,Y) which is true when X and Y are different numbers.*

Lecture five:

Repetition and Recursion

1. Repetition

2. Recursion

2.1 Tail recursion

2.2 Non-tail recursion

1. Repetition

In prolog there is a constant formula to generate repetition; this technique can generate repetition for some operation until the stopping condition become true.

Example: prolog program read and write a number of characters continue until the input character equal to ‘#’.

```
Predicates
Repeat.
Typewriter.

Clauses
Repeat.
Repeat:-repeat.
Typewriter:-repeat,readchar(C),write(C  ),nl,C=’#’,!.
```

2.Recursion

In addition to have rules that use other rules as part of their requirements, we can have rules that use themselves as part of their requirements.

This kind of rule called “recursive “because the relation ship in the conclusion appears again in the body of the rule, where the requirements are specified.

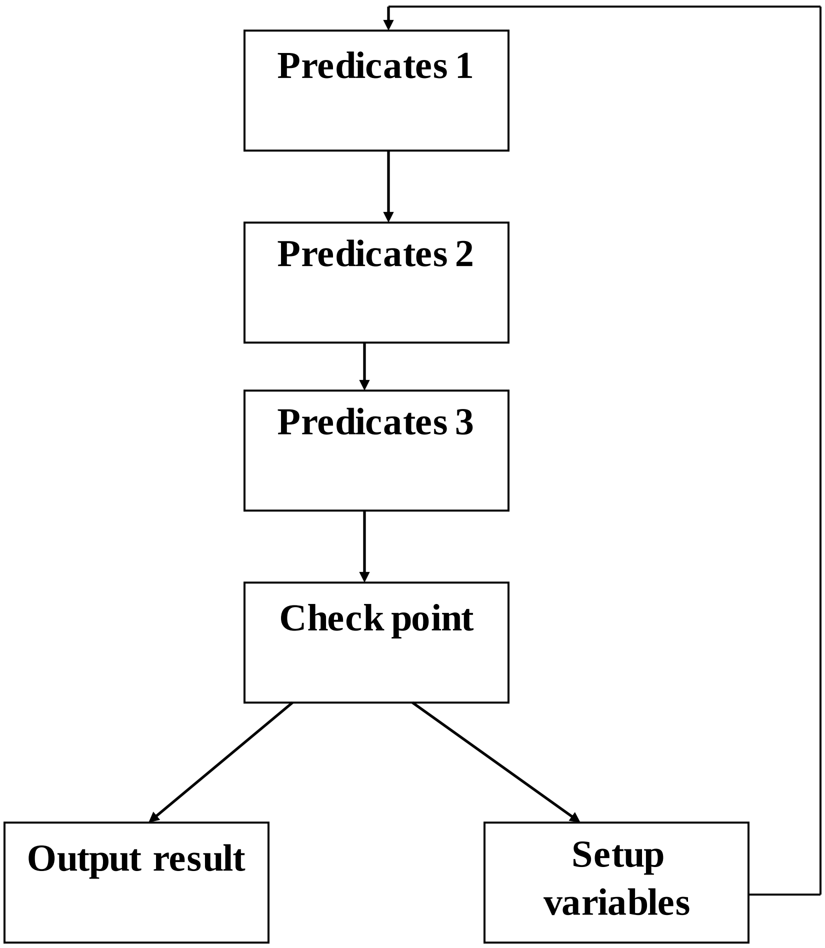
A recursive rule is a way of generating a chain of relationship for a recursive rule to be effective. However, there must be some place in the chain of relationship where the recursion stops.

This stopping condition must be answerable in the database like any other rule.

2.1 Tail Recursion

We place the predicate that cause the recursion in the tail of the rule as shown below:

Head :- p1,p2,p3, head.



Example 1: program to print number from n to 1.

```
Predicates
      A (integer)
Clauses
     A(1) :- write (1), nl ,!.
     A(M):-  write (M) , nl, M1 = M -1, A( M1).

Goal
A(4)
```

Output:

```
4
3
2
1
Yes
```

Example 2: program to find factorial.

5! = 5*4*3*2*1

```
Predicates
    Fact ( integer,  integer,  integer)
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.
```

Example 3: program to find power .

3^4 = 3*3*3*3

```
Domains
   I= integer
Predicates
  Power ( I,I,I, I ).
Clauses
  Power ( X,Y,P,R):-  P1= P*X, Y1 =Y-1, power(X,Y1,P1,R).
  Power (_,0,P,P):-!.

Goal
Power(3,2,1,P)

Output
P= 9
```

2.2 Non –Tail Recursion (Stack Recursion)

This type of recursion us 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.. Below a simple comparison between tail and non-tail recursion.

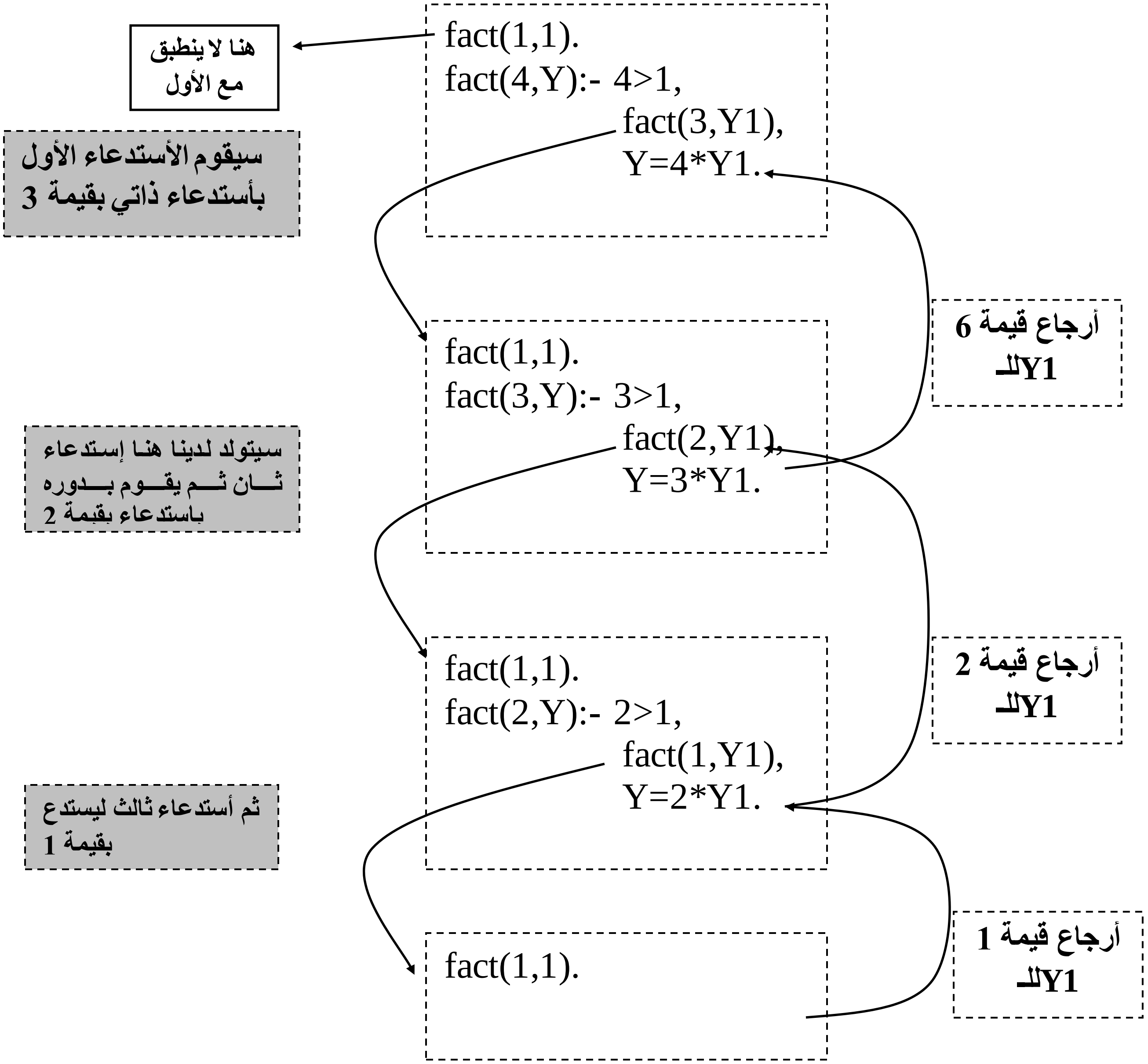
Tail recursion	Non-tail recursion
1. Call for rule place in the end of the rule. 2. It is not fast as much as stack recursion. 3. Use more variable than stack recursion.	1. Call for the rule place in the middle in the rule. 2. Stack recursion is fast to implement. 3. Few parameters are used.

Example 4: factorial program using non-tail recursion.

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



    Goal
    Fact (4,Y)

    Output:
    Y =24.
```

Example 5: power program using non-tail recursion.

```
Predicates
  Power (integer, integer, integer)
 Clauses
   Power (_,0,1):-!.
   Power ( X,Y,P) :- Y> 0, Y1=Y -1, power (X,Y1,P1),P= X*P1.
Goal
   Power (3,2,Z)

Output
    Z = 9.
```



هنا لا ينطبق
مع الأول

```
fact(1,1).
fact(4,Y):- 4>1,
            fact(3,Y1),
            Y=4*Y1.
```

سيقوم الأستدعاء الأول
بأستدعاء ذاتي بقيمة 3

```
fact(1,1).
fact(3,Y):- 3>1,
            fact(2,Y1),
            Y=3*Y1.
```

أرجاع قيمة 6
للـY1

سيتولد لدينا هنا إستدعاء
ثان ثم يقوم بدوره
باستدعاء بقبمة 2

```
fact(1,1).
fact(2,Y):- 2>1,
            fact(1,Y1),
            Y=2*Y1.
```

أرجاع قيمة 2
للـY1

ثم أستدعاء ثالث ليستدع
بقيمة 1

```
fact(1,1).
```

أرجاع قيمة 1
للـY1

H.W

1. Write prolog program to find the sum of 10 integer element using tail and non tail recursion.
2. Write prolog program to find the maximum value between 10 elements.
3. Write prolog program to find the minimum value between 10 elements.
4. *Find the sum S = 1+2 + 3 ….+N*

Lecture Six:

String standard predicates

1. ***Isname(string) test if the content of the string is name or not***

   Isname("abc") yes
   Isname("123"). No

2.char_int(char,integer) convert the character to its integer value and the opposit

```
                Char_int('A',X)
                      X=65
               Char_int(X,65)
                      X='A'
```

3. Str_char(string,char) convert the string (of one char) to char and the opposit

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

4. str_real (string,real) convert the string (ofreal) to real and the opposit

```
          Str_real("0.5",X)
                X=0.5
         Str_real(X,0.5)
               X="0.5"
```

5.Fronttoken(string,string,string).

Take token of word from the string and return the reminder of the string .

Fronttoken(string,token,rem).

```
      Fronttoken("ab cd ef",X,Y).
       X="ab"    y="cd ef"
      Fronttoken("c def",X,Y)
     X="cd"    Y="ef"
```

6. ***Frontstring(integer,string,string,string)***

Take a string(str) with length specified by the integer value and return the reminder

Frontstring(integer,string,str,rem)

```
    Frontstr(3,"ahmed",X,Y)
        X="ahm"    Y="ed"
   Frontstr(2,"abcde",X,Y).
       X="ab"    Y="cde"
  Frontstr(3,S,"ahm","ed").
      S="ahmed"
```

7. ***Frontchar(string,char,string).***
   Take one char from a specific string and return the reminder

```
   Frontchar(string,char,rem).
 Frontchar("ahmed",X,Y)
    X='a'   Y="hmed"
Frontchar(X,'a',"hmed")
     X="ahmed"
```

8. ***Str_len(string,length)***
   Return the length of specific string

Str_len("ahmed",X)

```
     X=5
 Str_len("ab",X)
   X=2
 Str_len("ab",3)    no
Str_len(X,3)     X="---"
```

9. ***Concat(string,string,string).***
   Concat two string together to produce one string

```
Concat("ab","cd",X)
      X="abcd"
```

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"
```

Prolog Programs that deal with string

Ex1:Pogram that read two string and concat them in one string as upper case.

```
predicates
start(string)
clauses
start(X):-readln(S),readln(S1),concat(S,S1,S2),upper_lower(X,S2).
```

Goal
Start(X)

Output:

```
Ahmed
Ali
X=AHMEDALI  yes
```

Ex2:program that read string of one character then return the integer value of this char.

```
predicates
start(integer).
clauses
start(X):-readln(S),str_char(S,X).
```

goal
start(X)

Output:

```
a
X=97
yes
```

Ex3: Program that take a string of words and print each word in a line as upper case.

```
predicates
start(string).
clauses

start(S):-fronttoken(S,S3,S2), upper_lower(S1,S3), write(S1),
nl,start(S2).
start("").
```

Goal
Start("ali is a good boy").

Output:

```
ALI
IS
A
GOOD
BOY
yes
```

Ex4: program that take a string and convert each character it contain to its corresponding integer value.

```
Predicates
start(string).
clauses

start(S):-frontstr(1,S,S1,S2), char_int(S1,I), write(I), nl , start(S2).
start("").
```

Goal
Start("abc").

Output:

```
97
98
99
Yes
```

Ex5: program that return the number of names in a specific string.

```
predicates
start(string,INTEGER).
clauses

start(S,X):-fronttoken(S,S1,S2),isname(S1),X1=X+1,start(S2,X1).
start(S,X):-fronttoken(S,_,S2),start(S2,X).
start("",X):-write("the  number of names is", X).
goal
start("ali has 2 cars").

Output:
 The no. of names is 3
Yes
```

Ex6:program that split a specific string to small string with length 3 char.

```
predicates
start(string).
clauses
start("").
start(S):-str_len(S,I),  I MOD 3=0, frontstr(3,S,S1,S2), write(S1),
nl,start(S2).
start(S):-concat(S," ",S1),start(S1).

Goal
Start("abcdefg").

Output:
abc
def
g
yes
```

H.W

1- *Write a prolog program that do the following: convert the string such as "abcdef" to 65 66 67 68 69 70.*

2-*Program tofind the number of tokens and the number of character in a specific string such as: "ab c def" the output is tokens and 6 character.*

Lecture Seven:

LIST

1. ***list in prolog***
2. ***syntax of list***
3. ***head and tail***

1. list in prolog

In prolog, a list is an object that contains an arbitrary number of other objects within it. Lists correspond roughly to array in other languages but unlike array, a list dose not require you to how big it will be before use it.

2. syntax of list

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

```
Domains
  list = integer*
```

- ‘*’ refer to list object which can be of length zero or un defined.
- The type of element list can be of any standard defined data type like integer, char … ect or user defined data type explained later.
- List element surrounded with square brackets and separated by comma as follow: l = [1, 2, 3, 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, if the list is non empty then this statement decompos the list into Head and tail otherwise (if the list is empty) this statement add element to the list.

4. list and recursion

As maintained previous list consist of many element, therefore to manipulate each element in the list we need recursive call to the list until it become empty.

Example 1: program to print list element in one line.

```
Domains
  L = integer*
Predicates
 Print (L)
Clauses
 Print ([ ]):-!.
 Print ( [ H|T]):- write (H) , print (T).

Goal
  Print ([1,4,6,8]).
Output:
   1468
```

Example 2: program to find sum of integer list.

```
Domains
 I= integer
L=i*

Predicates
 Sum ( L I, I)
Clauses

Sum ( [ ],S,S):-!.
Sum( [H| T],S1,S):- S2 = S1+H , Sum (T,S2,S).

Goal
Sum ( [ 1,4,6,9],0 ,S).

Output
S = 20
```

Example 3: prolog program to spilt list into to list positive and negative list.

```
Domains
L= integer*

Predicates
Spilt ( L,L,L)

Clauses

Spilt ( [ ],[ ],[ ]):-!.
Spilt ( [ H| T],[H|T1],L2):- H>= 0,!,spilt (T, T1,L2).
Spilt ([H|T],L1,[H|T2] ) :- spilt ( T,L1,T2).

Goal
Spilt ([-1,4,-9,8,0],L1,L2).
L1 = [4,9,0]
L2 = [-1,-9]
```

H.W

1. Write prolog program to find the union of two lists.

2. Write prolog program to find the intersection between two lists.

3. Write prolog program to find the difference between two lists.

4. Write prolog program that check the equality between two lists.

5. Write prolog program to find the last element in a list.

6. Write prolog program to find the union of two lists.

7. Write prolog program to find the length of a list.

8. Write prolog program to find the index of specified element in a list.

9. Write prolog program to get the element at nth index lists.

10. Write prolog program that replace specified element in a list with value 0.

11. Write prolog program that delete a specified element in a lists.

12. Write prolog program that take two lists as input and produce a third list as output, this list is the sum of the two lists.

13. Write prolog program that multiply each element in the list by 5.

14. Write prolog program that sort a list descending.

15. Write prolog program that convert any given decimal number to its binary representation and store it in a list.

Lecture Eight:

Database manipulation in prolog

Example1:

1- **Assert predicate:**

- assert(X) or assertz(X) :Adds a new fact to the database. Term is asserted as the last fact with the same key predicate.

  ✓ For example;

```
domains
s=string.
ls=s*.
database
person(s)
predicates
list_preson(ls)
clauses
list_preson(L):-
    assert(person ("Ali")),
    assert(person ("Zaki")),
    assert(person ("Suha")),
    findall(X,person(X),L).

goal: list_preson(L).
%L=["Ali","Zaki","Suha"]
```

- asserta(X) :Same as assert, but adds a fact X at the beginning of the database.

  ✓ For example;

```
domains
s=string.
ls=s*.
database
person(s)
predicates
list_preson(ls)
clauses
```

```
list_preson(L):-
        asserta(person ("Ali")),
        asserta(person ("Zaki")),
        asserta(person ("Suha")),
        findall(X,person(X),L).

goal: list_preson(L).
%L=["Suha","Zaki","Ali"]
```

2- **Retract predicate:**

- retract(X): Removes a fact X from the database.
  - ✓ For example;

```
domains
s=string.
ls=s*.
database
person(s)
predicates
list_preson(ls)
clauses
list_preson(L):-
        assert(person ("Ali")),
        assert(person ("Zaki")),
        assert(person ("Suha")),
        retract(person ("Zaki")),
        findall(X,person(X),L).

goal: list_preson(L).
%L=["Ali","Suha"]
```

- retractall(X): Removes all facts from the database for which the head unifies with X.
  - ✓ For example;

```
domains
```

```
s=string.
ls=s*.
database
person(s)
predicates
list_preson(ls)
clauses
list_preson(L):-
        assert(person ("Ali")),
        assert(person ("Zaki")),
        assert(person ("Suha")),
        retractall(person (_)),% retractall(_),
        findall(X,person(X),L).

goal: list_preson(L).
%L=[]
```

Example 2: Use a database concept to perform the following goal:
Goal: run("He bought 7 oranges their total weight 1.5 kg").
And give the following output:

```
String=  He      length=  2
String=  bought     length=  6
String=  oranges     length=  7
String=  their     length=  5
String=  total     length=  5
String=  weight     length=  6
String=  kg     length=  2
```

Solution:

```
database
db_string(String,integer)
predicates
split_tokens(string)
run(string)
print_string
```

```
clauses
run(S):-retractall(_),  split_tokens(S),  print_string.

split_tokens(S):-
   fronttoken(S,W,R),   isname(W),!,str_len(W,N),  assert(db_string(W,N)),
   split_tokens(R).
split_tokens(S):-   fronttoken(S,_,R),!,split_tokens(R).
split_tokens("").

print_string:-
db_string(S,N),write("String=   ",S,"      length=   ",N),nl,fail.
print_string.

goal
    run("He bought 7 oranges  their total  weight 1.5 kg").
/*String=   He      length=   2
String=   bought      length=   6
String=   oranges      length=   7
String=   their      length=   5
String=   total      length=   5
String=   weight      length=   6
String=   kg      length=   2
yes
 */
```

Lecture nine:

File manipulation in prolog

1- Openread (file name, symbolic file name).
openread(file,”d:\\f1.pro”)

2- Openwrite (file name, symbolic filename)
openwrite(file,”d:\\f1.pro”)

3- Readdevice is used to read from file or from the keyboard.
readdevice(file)
readdevice(keyboard)

4- writedevice: is used to write text to a file or on the screen
Writedevice(file)
Writedevice (screen)

5-existfile(symbolic filename): is used to chec if the file is available or not.
existfile(“d:\\f1.pro”)

6-eof(filename): is used to check the end of the file.
eof(file)

7-file_str(symbolic filename,string): is used to chane the content of the file to string.
file_str(“d:\\f1.pro”,S)

Example1: program can saving any string in a file and printing it.

```
domains
   file=m
   s=string
predicates
   readfile(s)
   writefile(s)
   start
```

```
clauses
start:-
   write("Save any string in your file(D:\\test.pro):"),nl,
   readln(X),
   writefile(X),
   readfile(Y),
   write("The string in your file is:",Y),nl.
writefile(X):-
   openwrite(m,"D:\\test.pro"),
   writedevice(m),
   write(X),
   closefile(m).
readfile(X):-
   openread(m,"D:\\test.pro"),
   readdevice(m),
   readln(X),
   closefile(m).
goal:start.
    %Save any string in your file(D:\test.pro):
    %abcdefg
    %The string in your file is:abcdefg
```

Example2 (a): Given a program to obtain a digit and its item from a string as shown below:

```
domains
    s=string
    i=integer
predicates
    digit(s,i,s)
    change(s,i)
clauses
    digit(S,D,I):-
         fronttoken(S,W,R),change(W,D),
         fronttoken(R,I,_).
    digit(S,D,I):-
        fronttoken(S,_,R),digit(R,D,I).
    change("five",5).
    change("ten",10).
Goal:
    digit("I have five pens and ten books.", Digit, Item).
 %Digit =5, Item=pens
 %Digit =10, Item=books
```

Example2 (b): Preform the above problem using file concept.

```
domains
    file = myfile
    s=string
    i=integer
predicates
    digit(s,i,s)
    change(s,i)
    writefile
    readfile(s)
    chk_digit(i,s)
clauses
writefile:-
   openwrite(myfile,"d:\\f1.pro"),
   write("Enter sentence:"),nl,
   readln(S),
   writedevice(myfile),
   write(S),
   closefile(myfile).

readfile(S):-
    openread(myfile,"d:\\f1.pro"),
    readdevice(myfile),
    readln(S),
    closefile(myfile).

chk_digit(D,I):-
    readfile(S),
    write("Source sentence is:\n",S),nl,
    write("The result is:\n"),
    digit(S,D,I).

    digit(S,D,I):-
         fronttoken(S,W,R),change(W,D),
          fronttoken(R,I,_).
    digit(S,D,I):-
         fronttoken(S,_,R),digit(R,D,I).
    change("five",5).
    change("ten",10).
goal
     writefile,chk_digit(Digit,Item).
/*Enter sentence:
I have five pens and ten books.
Source sentence is:
I have five pens and ten books.
The result is:
Digit=5, Item=pens
Digit=10, Item=books
2 Solutions*/
```

APPENDIX A/ SOME PROLOG EXAMPLES ABOUT ITEMS

```
TRACE
domains
S=SYMBOL

predicates
son(s,s)
father(s,s).
brother(s,s).
cousin(s,s).
grandfather(s,s).
uncle(s,s).

clauses
son(ali,ahmed).
son(hamza,ahmed).
son(ahmed,majed).
son(hassan,majed).
son(hussain,hassan).

father(X,Y):-son(Y,X).
brother(X,Y):-father(Z,X),father(Z,Y),X<>Y.
cousin(X,Y):-father(Z,X),father(W,Y),brother(Z,W).
grandfather(X,Y):-father(X,Z),father(Z,Y).
uncle(X,Y):-father(Z,Y),brother(X,Z).
```

```
/* Draw the the following shape
                                *
                                * *
                                * * *
                                * * * *
                                * * * * *
domains
i=integer
predicates
star(i).
print(i).

clauses
star(6).
star(I):-print(I),I1=I+1,star(I1).
print(0):-nl.
print(J):-write(' ','*'),J1=J-1,print(J1).
goal
star(0).
```

Prolog program to read and print an input item

```
predicates
print.
clauses
print:-write("please read integer number\n"),readint(X),
WRITE("YOU READ  ",X),nl.
```

Prolog program to test the input number is odd or even

```
domains
i=integer
predicates
odd_even(i).
clauses
odd_even(X):-X mod 2=0,write("even \n").
odd_even(_):-write("odd\n").
```

Prolog program to test the input number is positive or negative

```
domains
i=integer
predicates
pos_neg(i).
clauses
pos_neg(X):-X>=0,write("positive \n").
pos_neg(_):-write("negative\n").
```

Prolog program to test the input number is prime or not.

```
domains
i=integer
predicates
prime(i,i).
clauses
prime(C,X):-C1=C+1,X<>C1,X mod C1<>0,prime(C1,X).
prime(C,X):-C1=C+1,X<>C1,write("not prime").
prime(_,_):-write("prime").
```

Appendix B / SOME PROLOG EXAMPLES ON LIST

```
1- /*append two list in one list*/
 /*domains
     d=integer.
      l=d*
 predicates
      app(L,L,L).
 clauses
       app([],L,L).
    app([H|L1],L2,[|L3]):-app(L1,L2,L3).

  goal
  app([1,2,3],[4,5,6],L).*/
```

```
2-  /* append three list in one list*/
/*domains
i=integer
s=i*
predicates
app(s,s,s,s)
clauses
app([],[],L,L).
app([H|T],L2,L3,[H|T1]):-app(T,L2,L3,T1),!.
app([],[H|T],L3,[H|T1]):-app([],T,L3,T1),!.

goal
app([1,2,3],[4,5,6],[7,8,9],L).*/
```

```
 3-  /* program to find the difference between two list such as
diff([1,2,3],[3,4,5],L),   L=[1,2].  */

/*domains

i=integer
l=i*
predicates
diff(l,l,l).
member(i,l).
clauses
diff([],_,[]):-!.
diff([H|T],Y,[H|T1]):-not(member(H,Y)),diff(T,Y,T1),!.
diff([_|T],Y,Z):-diff(T,Y,Z).
member(X,[X|_]):-!.
member(X,[_|T]):-member(X,T).
goal
 diff([1,2,3],[3,4,5],L). */
 4-  /* program to union two lists of charecter in one list with
discard the iteration charecter such as:
union(['a','b','d'],['c','d','e'],L)  , L=['a','b','c','d','e'].

or  union(['c','d','e'],['f','c'],L)  ,     L=['d','e','f','c']
*/
/*domains
j=char
c=j*

predicates
union(c,c,c).
mem(j,c).

clauses
union([],[],[]):-!.
union(X,[],X):-!.
union([],X,X):-!.
union([H|T],L,[H|T1]):-not(mem(H,L)),!,union(T,L,T1).
union([_|T],L,L1):-union(T,L,L1).

mem(H,[H|_]):-!.
mem(X,[_|T]):-mem(X,T).
goal
union(['c','d','e'],['f','c'],L).*/
```

```
  5- /* program to find the Intersection between two lists of
charecter in one list with discard the iteration charecter such
as:  intersect(['a','b','d'],['c','d','e'],L)  , L=[].

or  intersect(['a','c','d','e'],['f','c'],L)  ,     L=['a','c']
*/
/*domains
j=char
c=j*

predicates
intersect(c,c,c).
mem(j,c).

clauses
intersect([],[],[]):-!.
intersect(_,[],[]):-!.
intersect([],_,[]):-!.
intersect([H|T],L,[H|T1]):-mem(H,L),!,intersect(T,L,T1).
intersect([_|T],L,L1):-intersect(T,L,L1).

mem(H,[H|_]):-!.
mem(X,[_|T]):-mem(X,T).
goal
intersect(['a','c','d','e'],['a','f','c'],L).*/
```

6- /* program to print list of integer number */

```
/*domains

i=integer
l=i*
predicates
print(l).
clauses
print([]).
print([H|T]):-write(H),write("-"),print(T).

goal
print([1,2,3]).*/
```

7- /*program to print list of symbols in reverse order such as L=[a,b,c,d]   L1=[d,c,b,a] */

```
/*domains
l=symbol*
predicates
writelist(l).

clauses
writelist([]).
writelist([H|T]):-writelist(T),write(H),nl.

goal
writelist([a,b,c,d]).*/
```

8- /* program to print list of character lists such as list([['a','b','c'],['d','e','f']]) the result is:- abcdef   */

```
/*domains
c=char
s=c*
s1=s*
predicates
list(s1)
clauses
list([[]]).
list([[H|T]]):-write(H),list([T]).
list([H|T]):-list([H]),list(T).

goal
list([['a','1','c'],['d','e','5']]).*/
```

9- /* program to delete item from a list */

```
/*domains
i=integer
l=i*
predicates
del(i,l,l).
clauses
del(_,[],[]):-!.
del(X,[X|T],T):-!.
del(X,[H|T],[H|T1]):-del(X,T,T1).

goal
del(3,[1,2,3,4],L).*/
```

10- /* program to find the value of an item in a list using its position */

```
/*domains
i=integer
l=i*

predicates
find(l,i,i,i).

clauses
find([],_,C,C):-!.
find([H|_],X,X,H):-!.
find([_|T],X,X1,H):-X2=X1+1,find(T,X,X2,H).

goal
find([1,2,3,4],3,0,H).*/
```

11- /* Program to delete a specific item from a list according to its position*/

```
/*domains
i=integer
s=i*
predicates
ins(i,i,s,s)
clauses
ins(X,0,Y,[X|Y]).
ins(X,D,[H|T],[H|T1]):-D1=D-1,ins(X,D1,T,T1).
goal
ins(X,0,Y,[1,2,3]).*/
```

12- /*program return the position of specific item in a list such as pos([5,8,0,9],8,POS), POS=2*/

```
/*domains
i=integer
s=i*
predicates
```

```
pos(s,i,i)
clauses
pos([Y|_],Y,1).
pos([_|T],Y,X):-pos(T,Y,X1),X=X1+1.

goal
pos([5,8,0,9],8,POS).*/


    13-  /* Program to reverse the item of list such as
rev([1,2,3,4],REV), REV=[4,3,2,1].  */

/*domains
i=integer
s=i*
predicates
rev(s,s)
app(s,s,s)
clauses
app([],X,X).
app([H|T1],X,[H|T]):-app(T1,X,T).
rev([X],[X]).
rev([H|T],L):-rev(T,L1),app(L1,[H],L).

goal
rev([1,2,3,4,8,0],REV).*/

   14-   /* find the summation of items in a list */
/*domains
i=integer
l=i*
predicates
sum(l,i).
clauses
sum([],0).
sum([H|T],S):-sum(T,S1),S=S1+H.

goal
sum([1,23,4],S).*/


   15-   /* Program to split list to two lists , the first list
has number of items as specified and the other has the reminder
of orginal list such as
                                      split(3,[5,8,9,7,5],L1,L2).
THE FIRST THREE ELEMENT GO TO L1 AND THE REMINDER TO L2 */
/*domains
i=integer
s=i*
predicates
split(i,s,s,s)
clauses
```

```
split(0,X,[],X).
split(C,[H|T],[H|T1],W):-C1=C-1,split(C1,T,T1,W).
goal
split(3,[5,8,9,7,5],L1,L2).*/
```

16- /* sort list in descending order using max and delete rules */

```
/*domains
i=integer
s=i*
predicates
max(s,i,i)
del(i,s,s)
sort(s,s)
clauses
max([],M,M).
max([H|T],X,M):-H>X,max(T,H,M),!.
max([_|T],X,M):-max(T,X,M).
del(_,[],[]).
del(X,[X|T],Y):-del(X,T,Y).
del(X,[H|T],[H|T1]):-del(X,T,T1),!.
sort([],[]).
sort(Y,[M|T1]):-max(Y,0,M),del(M,Y,U),sort(U,T1),!.

goal
sort([5,4,9,7,3],L).*/
```

17- /* sort in ascending order using insert rule */

```
/*domains
i=integer
l=i*

predicates
insert(i,l,l).
sort(l,l).

clauses
insert(Y,[H|T],[H|T1]):-Y>H,!,insert(Y,T,T1).
insert(H,T,[H|T]).
sort([],[]):-!.
sort([H|T],L):-sort(T,T1),insert(H,T1,L).

goal
sort([11,9,8,0],L).*/
```

18- /*program to read students marks and find the average of each student ,then return the max degree and the min degree and sort the averages in ascending order

```
domains
i=real
state=st(i,i).
l=state*
li=i*

predicates
s_avg(l,li).
start(l,li).
max(li).
min(li).
sort(li,li).
insert(i,li,li).
maxx(l,li).
minn(l,li).
run(l).

clauses
s_avg([],[]).
s_avg([st(H,H1)|T],[L1|L]):-L1=(H+H1)/2,s_avg(T,L),nl.


start(S,FIN):-s_avg(S,FIN),write(FIN),nl,!,run(S),
             write("the sort
list"),sort(FIN,FIN1),!,write(FIN1),nl.

run(S1):-write(" the max mark is:"),maxx(S1,L),max(L),nl,
         write(" the min mark is:"),minn(S1,L1),min(L1),nl.

                  find the minimum  item in a alist such as :
max([22,15,14]) max=22 /
max([]):-!.
max([H]):-write(H).
max([H,Y|T]):-Y>H,max([Y|T]).
max([H,Y|T]):-H>Y,max([H|T]).
                  find the minimum  item in a alist such as :
Min([22,15,14]) min=14
min([]):-!.
min([H]):-write(H).
min([H,Y|T]):-Y<H,min([Y|T]).
min([H,Y|T]):-H<Y,min([H|T]).

            sort in ascending order using insert rule
insert(Y,[H|T],[H|T1]):-Y>H,!,insert(Y,T,T1).
insert(H,T,[H|T]).
sort([],[]):-!.
sort([H|T],L):-sort(T,T1),insert(H,T1,L).
            find the max number in one state  such as :
maxx([st(22,18),st(45,88),st(11,19)],L) , L=[22,88,19]
maxx([],[]):-!.
maxx([st(H,H1)],[H]):-H>H1.
maxx([st(H,H1)],[H1]):-H1>H.
maxx([st(H,H1),st(T1,T2)|T],[H|M]):-H>=H1,maxx([st(T1,T2)|T],M).
maxx([st(H,H1),st(T1,T2)|T],[H1|M]):-H1>=H,maxx([st(T1,T2)|T],M).
           find the min number in one state  such as :
minn([st(22,18),st(45,88),st(11,19)],L) , L=[18,45,11]
minn([],[]):-!.
minn([st(H,H1)],[H]):-H<H1.
minn([st(H,H1)],[H1]):-H1<H.
minn([st(H,H1),st(T1,T2)|T],[H|M]):-H<=H1,minn([st(T1,T2)|T],M).
minn([st(H,H1),st(T1,T2)|T],[H1|M]):-H1<=H,minn([st(T1,T2)|T],M).

goal
start([st(66,85),st(23,56),st(77,67)],AVG).*/
```

19- /* program to split the list to two lists , the first has the odd integer value and the second has the even integer value */

```
/*domains
i=integer
l=i*.

predicates
plist(i,l,l,l).

clauses

plist(_,[],[],[]):-!.
plist(N,[H|T],[H|T1],L):-N1=N+1,N1 mod 2<>0,!,plist(N1,T,T1,L).
plist(N,[H|T],L2,[H|T2]):-N1=N+1,plist(N1,T,L2,T2).

goal
plist(0,[1,2,3,4],L,L1).*/
```

20- /* read a string then find how many integer numbers and real numbers in the string save the integer number in list1 and the real number in list2 */

```
/*domains
s=string
l=s*
i=integer
l1=i*
r=real
l2=r*
predicates
split(l,l1,l2,i,i,i,i).
clauses
split([],[],[],I,I,R,R).
split([H|T],[H1|T1],L2,I1,I,R1,R):-
fronttoken(H,X,Y),Y="",str_int(X,Z),H1=Z,I2=I1+1,!,split(T,T1,L2,
I2,I,R1,R).
split([H|T],L1,[H2|T2],I1,I,R1,R):-
str_real(H,Z),H2=Z,R2=R1+1,!,split(T,L1,T2,I1,I,R2,R).


goal
split(["192","3.3","99","34.4"],L1,L2,0,I,0,R).*/

  21-  /* Program find the iterations of integer value in a list
*/
/*domains
i=integer
l=i*

predicates
match(i,i,i).
iteration(i,i,l).

clauses
match(X,X,1).
match(X,Y,0):-X<>Y.

iteration(0,_,[]).
iteration(S,X,[Y|Z]):-match(Y,X,Q),iteration(S1,X,Z),S=S1+Q.

goal
iteration(I,2,[1,2,3,2,3,4,3]).*/


    22-  /* delete  the first N item from a list such as
del(3,[1,2,3,4,5],L) , L=[4,5].  */
/*domains
i=integer
l=i*

predicates
del(i,l,l).

clauses
del(_,[],[]):-!.
del(0,X,X):-!.
del(N,[_|T],L):-N1=N-1,del(N1,T,L).

goal
del(3,[1,2,3,4,5],L).*/


   23- /* delete  the last N item from a list such as
del(3,[1,2,3,4,5],L) , L=[1,2].  */
/*domains
```

```
i=integer
l=i*

predicates
del(i,l,l).
len(l,i).
copy(i,l,l).

clauses
del(N,L,F):-len(L,M),N1=M-N,copy(N1,L,F).

len([],0).
len([_|T],LEN):-len(T,LEN1),LEN=LEN1+1.

copy(_,[],[]):-!.
copy(N,_,[]):-N=0,!.
copy(N,[H|T],[H|Z]):-N1=N-1,copy(N1,T,Z).


goal
del(7,[1,2,3,4,5,6,7,8,9],L).*/

   24-  /* program to rotate the items of a list to the left by N
location. such as:- rotate(1,[1,2,3],L),  L=[2,3,1].  */

/*domains
i=integer
l=i*

predicates
shift(l,l).
append(i,l,l).
rotate(i,l,l).

clauses
rotate(_,[],[]):-!.
rotate(0,X,X):-!.
rotate(N,X,Z):-N1=N-1,shift(X,Y),rotate(N1,Y,Z).

shift([H|T],Z):-append(H,T,Z).

append(X,[],[X]):-!.
append(X,[H|T],[H|T1]):-append(X,T,T1).

goal
rotate(3,[1,2,3,4,5],L).*/

  25-/* program to square the items of a list of integer value*/

/*domains
i=integer
l=i*
```

```
predicates
sqr(l,l).

clauses

sqr([],[]).
sqr([H|T],[H1|T1]):-H1=H*H,sqr(T,T1).

goal
sqr([1,2,3,4],L).*/

   26- /* program to find if a list is a subset of another list
such as : subset([1,2,3],[5,6,1,9,2,3,4,5])---> yes

subset([1,2,3],[5,6,9,2,3,4,5])-----> no */
/*domains
i=integer
l=i*

predicates
subset(l,l).
append(l,l,l).

clauses
subset([],_):-!.
subset([H|T],Z):-
append(L1,[H|L2],Z),append(L1,L2,L3),subset(T,L3).

append([],Y,Y):-!.
append([H|T],Y,[H|Z]):-append(T,Y,Z).

goal
subset([1,2,3],[5,6,1,9,2,3,4,5]).*/
```

27- Prolog program to convert any decimal number to it binary

```
DOMAINS
I=INTEGER
L=INTEGER*
predicates
sum(I,l).
print(l).

clauses
sum(0,[0]):-!.
sum(1,[0,1]):-!.
sum(X,[H|T]):- H= X MOD 2,X1= X DIV 2,sum(X1,T).
print([]).
print([H|T]):-print(T),write(H).
```