

Experiment No. 01

Experiment Name: Introduction to SWI –Prolog Programming with the help of simple programs.

Objective:

- To understand logical programming syntax and semantics.
- To Design program in prolog language.

Theory:

Prolog has its roots in first-order logic, a formal logic, and unlike many other programming languages, Prolog is declarative: the program logic is expressed in terms of relations, represented as facts and rules. A computation is initiated by running a query over these relations. Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching databases, voice control systems, and filling templates.

Rules and facts:

Prolog programs describe relations, defined by means of clauses. Pure Prolog is restricted to Horn clauses. There are two types of clauses: facts and rules. A rule is of the form

Head:- Body.

and is read as "Head is true if Body is true". A rule's body consists of calls to predicates, which are called the rule's goals. The built-in predicate, /2 (meaning a 2-arity operator with name,) denotes conjunction of goals, and; /2 denotes disjunction. Conjunctions and disjunctions can only appear in the body, not in the head of a rule.

Clauses with empty bodies are called facts. An example of a fact is:

cat (tom).

Simple programs

Write a prolog program that describes a family's gene logical relationships.

Theory:

At first, we will define each member's identity in the family. Then we will define the relationship among them.

Program Syntax:

The following facts and rules are defined:

Facts:

- 1) female(X).=>Defines female.
- 2) male(Y).=>Defines male.
- 3) parentof(A,X/Y).=>Defines parent of X or Y.

Rules:

1) X and Y are sibling if Z is parent of X and Y. `siblingof(X,Y):- parentof(Z,X),parentof(Z,Y).`

2) X is brother of Y if Z is parent of X and Y and X is male. `brotherof(X,Y):- parentof(Z,X),male(X),parentof(Z,Y).`

Code:

`female(amy).`

`female(johnette).`

`male(anthony).`

`male(bruce).`

`male(ogden).`

`parentof(amy,johnette).`

`parentof(amy,anthony).`

`parentof(amy,bruce).`

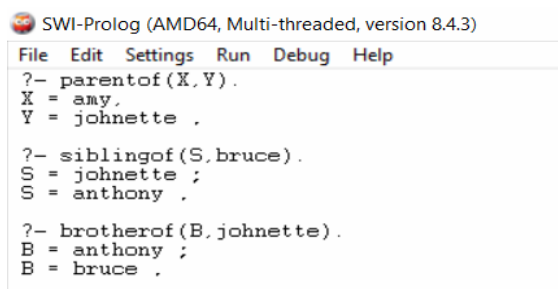
`parentof(ogden,johnette).`

`parentof(ogden,anthony).`

`parentof(ogden,bruce).`

`siblingof(X,Y) :- parentof(Z,X), parentof(Z,Y).`

`brotherof(X,Y) :- parentof(Z,X), male(X), parentof(Z,Y).`

Query and Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- parentof(X,Y).
X = amy,
Y = johnette ,

?- siblingof(S,bruce).
S = johnette ;
S = anthony ,

?- brotherof(B,johnette).
B = anthony ;
B = bruce ,
```

Prolog program to print a list of items.

Theory: There are some product items. We will print the items using prolog.

Program Syntax: The following facts and rules are defined:

Facts: Define item.

Item(X).

Rules: Write the item names.

products:-item(X),write(X),nl,fail.

Here, nl=Provides a new line on the screen.

write=Used to write the sum on the screen.

fail=immediately fail when prolog encounters it as a goal.

Code:

item(tshirt).

item(pant).

item(caps).

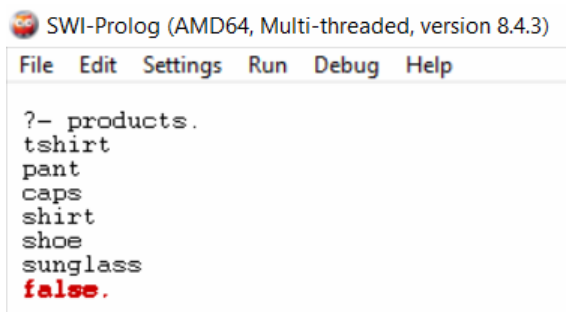
item(shirt).

item(shoe).

item(sunglass).

products:- item(X),write(X),nl,fail.

Query and Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- products.
tshirt
pant
caps
shirt
shoe
sunglass
false.
```

Prolog program to find maximum between two numbers.

Objective:

- To learn to find maximum of two numbers in prolog.

Theory:

We will compare two numbers using logical operator, \geq and $>$.

If X is greater than Y, then X is maximum, otherwise y is maximum.

Rules:


1) If X is greater than Y, then X is maximum $\text{max}(X,Y,X):-X \geq Y$.

2) If Y is greater than X, Then Y is maximum. $\text{max}(X,Y,Y):-Y > X$.

Code:

$\text{max}(X,Y,X):-X \geq Y$.

$\text{max}(X,Y,Y):-Y > X$.

Query and Output: SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File	Edit	Settings	Run	Debug	Help
?- max(10,20,P) . P = 20 .					
?- max(25,50,MAX) . MAX = 50 .					
?-					

Prolog program to find the GPA of a student.

Objective:

- To learn to find GPA in prolog programming.

Theory:

At first, we will define GPA of some students with their name and GPA. Then we will find GPA of a student by entering his/her name and print the GPA.

Fact:

X = student name, Y= GPA of the student. `gpa(X,Y).`

Rules:

Read student name from user and write the GPA. `read(X),gpa(X,Y),nl, write(Y).`

Code:

```
gpa(ashik,3.45).
```

```
gpa(rukon,3.50).
```

```
gpa(moni,3.25).
```

```
gpa(zaman,3.35).
```

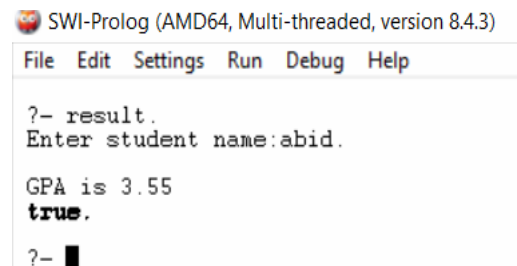
```
gpa(abid,3.55).
```

```
result:- write('Enter student name:'),
```

```
        read(X),gpa(X,Y),nl,
```

```
        write('GPA is '),
```

```
        write(Y).
```

Query and Output:

The screenshot shows the SWI-Prolog (AMD64, Multi-threaded, version 8.4.3) interface. The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main text area contains the following text: `?- result.`, `Enter student name:abid.`, `GPA is 3.55`, `true.`, and `?- █`.

A Prolog program that describes the student-professor relationships.

Objective: To learn about predicates in prolog programming.

Theory: At first, we will define each member's identity. Then we will define the relationship among them.

Program Syntax: The following facts and rules are defined:

Facts:

- 1) studies(X,Y).=>Defines students and courses.
- 2) teaches(X,Y).=>Defines teachers and courses.
- 3) professor(A,X/Y).=>Defines professor of X or Y.

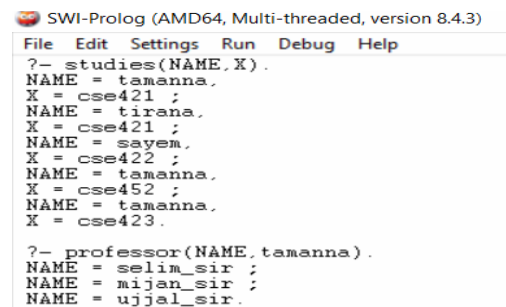
Rules:

- 1) X is professor of Y if X teaches Z and Y studies Z. professor(X,Y):- teaches(X,Z),studies(Y,Z).

Code:

```
studies(tamanna,cse421).  
studies(tirana,cse421).  
studies(sayem,cse422).  
studies(tamanna,cse452).  
studies(tamanna,cse423).  
teaches(selim_sir,cse452).  
teaches(mijan_sir,cse421).  
teaches(ujjal_sir,cse423).  
professor(X,Y):- teaches(X,Z),studies(Y,Z).
```

Query and Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
?- studies(NAME,X).  
NAME = tamanna,  
X = cse421 ;  
NAME = tirana,  
X = cse421 ;  
NAME = sayem,  
X = cse422 ;  
NAME = tamanna,  
X = cse452 ;  
NAME = tamanna,  
X = cse423 .  
  
?- professor(NAME,tamanna).  
NAME = selim_sir ;  
NAME = mijan_sir ;  
NAME = ujjal_sir .
```

A prolog program to find nth number of Fibonacci series.

Objective: To learn how to find nth number of Fibonacci series in prolog programming.

Theory: The Fibonacci numbers, commonly denoted F_n , form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is, $F_0 = 0$, $F_1 = 1$, And $F_n = F_{n-1} + F_{n-2}$ For $n > 1$.

Program Syntax:

The following facts and rules are defined:

- 1) 1st number of Fibonacci series is defined as 1. `fib(1,1).`
- 2) 2nd number of Fibonacci series is defined as 1. `fib(2,1).`
- 3) Nth number (where, $N > 2$) of fibonacci series is defines as: `fib(N,F):- N1=N-1, N2=N-2, fib(N1,R1), fib(N2,R2), R=R1+R2.`

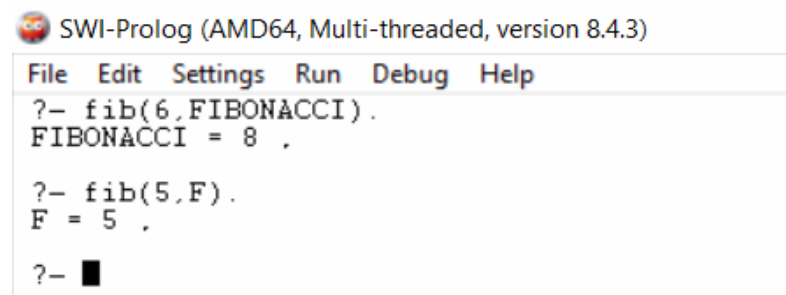
Code:

`fib(1,1).`

`fib(2,1).`

```
fib(N,F):- N>2,
    N1 is N-1,
    N2 is N-2,
    fib(N1,F1),
    fib(N2,F2),
    F is F1+F2.
```

Query and Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- fib(6,FIBONACCI).
FIBONACCI = 8 .

?- fib(5,F).
F = 5 .

?-
```

A prolog program to calculate factorial of a number.

Objective:

To know to calculate factorial of a number using prolog programming.

Theory:

We know, factorial of a number n is, $n! = n \times (n - 1) \times (n - 2) \times \dots \times 1$

Program Syntax:

To calculate factorial, the following facts and rules are defined:

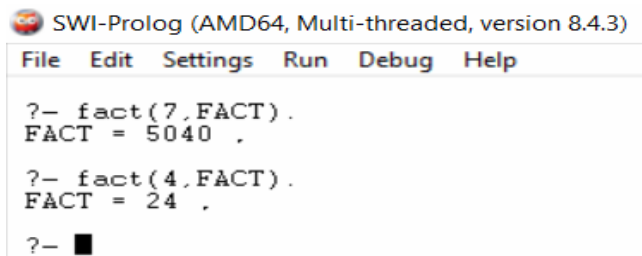
- 1) If $N=0$, then $R=1$. i.e., $0! = 1$. `fact(0,1).`
- 2) If $N \neq 0$, $N! = R$:- call `fact(N1,R1)`, $N=N1+1$, $R=R1*N$.

Code:

```
fact(0,1).
```

```
fact(N,F):- N>0,  
            N1 is N-1,  
            fact(N1,F1),  
            F is N*F1.
```

Query and Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
?- fact(7,FACT).  
FACT = 5040 .  
?- fact(4,FACT).  
FACT = 24 .  
?-
```

Conclusion:

We have successfully understand logical programming syntax and semantics. Program designing in prolog language with the help of some simple programs.

.

Experiment No:02

Experiment Name: Introduction to prolog programs rules, meaning with the help of some programs.

Objective:

- To understand prolog programs rules, meaning with the help of some programs.

Some Programs:

A prolog program to demonstrate the cube of a number.

Objective:

- Learn about input and output in prolog programming.

Theory:

We will read input from keyboard and write the output. We will demonstrate the cube of a number, $n^3 = n * n * n$. Here, read(X): is used for reading terms from current input stream. Write(X): is used for outputting term X on the current output file.

Rules:

Read X from keyboard and calculate cube and write the output: cube:-read(X), process (X).

process(N):-C=N*N*N, write (C),cube.

Code:

```
cube:-    write('Enter a number:'),
          read(X),process(X).
process(stop) :- !.
process(N) :- C is N*N*N,
              write('Cube of '), write(N),
              write('is '),write(C),nl, cube.
```

Query and Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- cube.
Enter a number: 3.
Cube of 3 is 27
Enter a number: |: stop.

true.
?- ■
```

A prolog program to satisfy a list of goals.

Objective: To learn how to satisfy a list of goals in prolog programs.

Theory: To satisfy a list of goals, at first, we will define some facts. Then we will form some rules using these facts. Then we will enter query to check if some goals are satisfied or not.

Facts:

- 1) Define big animals big(X).
- 2) Define small animal. small(X).
- 3) Define brown animal. brown(X).
- 4) Define black animal. black(X).
- 5) Define gray animal. gray(X).

Rules:

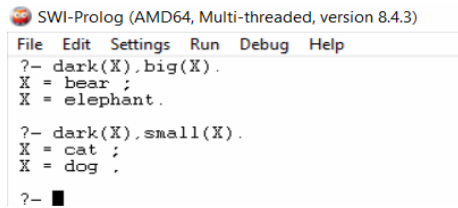
- 1) Anything black is dark. dark(Z):-black(Z).
- 2) Anything brown is dark. dark(Z):-brown(Z).
- 3) Anything brown is dark. dark(Z):- gray(Z).

Code:

```
big(bear).big(elephant).
small(cat).small(dog).
brown(bear).
black(cat).black(dog).
gray(elephant).
dark(Z) :- black(Z).
dark(Z) :- brown(Z).
```

dark(Z) :- gray(Z).

Query and Output:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
?- dark(X),big(X).
X = bear ;
X = elephant .

?- dark(X),small(X).
X = cat ;
X = dog .

?-
```

A prolog program to find GCD and LCM of two numbers.

Objective:

- To learn to find the GCD and LCM of two numbers in prolog programming.

Theory:

GCD: The greatest common divisor (gcd) of two or more integers, which are not all zero, is the largest positive integer that divides each of the integers. For two integers x, y, the greatest common divisor of x and y is denoted gcd(x,y).

LCM: The least common multiple of two integers a and b, usually denoted by lcm(a, b), is the smallest positive integer that is divisible by both a and b.

Program Syntax: Following facts and rules are defined:

- 1) If the two numbers are X and 0, then gcd=X. gcd(X,0,X).
- 2) If the two numbers are X and Y, then R=X mod Y, gcd(Y,R,D) gcd(X,Y,D):- R is X mod Y, gcd(Y,R,D). 3) If the two numbers are X and Y, gcd(X,Y,D), M=(X*Y)/D. lcm(X,Y,M):- gcd(X,Y,D),M is (X*Y)/D.


Code:

gcd(X,0,X).

gcd(X,Y,D):- R is X mod Y, gcd(Y,R,D).

lcm(X,Y,M):- gcd(X,Y,D),M is (X*Y)/D.

Query and Output:

 SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)

File Edit Settings Run Debug Help

```
?- gcd(12,9,GCD).  
GCD = 3 ,
```

```
?- lcm(5,7,LCM).  
LCM = 35 ,
```

Conclusion: We have understood prolog programs rules, meaning with the help of some programs.

Experiment no:3

Experiment name: Prolog programs on list.

Objective: To understand how the list is used in prolog programming.

Theory:

The list is a simple data structure that is widely used in non-numeric programming. List consists of any number of items, for example, red, green, blue, white, dark. It will be represented as, [red, green, blue, white, dark]. The list of elements will be enclosed with square brackets.

A list can be either empty or non-empty.

For example:

An empty list: L[[]].

A not-empty list: L[a, b, c, d].

A not-empty list can be represented as Head and Tail. If we consider the list L[a, b, c, d] then head of the list will be a and tail will be b, c, d. Can be represented as L[a|Tail] where Tail=[b, c, d].

To learn the list in prolog programming, we need to understand following topics about list:

1. Basic Operations on Lists like insert, delete etc.
2. Set operations like set union, set intersection, etc.

Basic Operations on Lists

Program for membership

```
list_member(X, [X|_]).  
list_member(X, [_|TAIL]) :- list_member(X, TAIL).
```

Query:

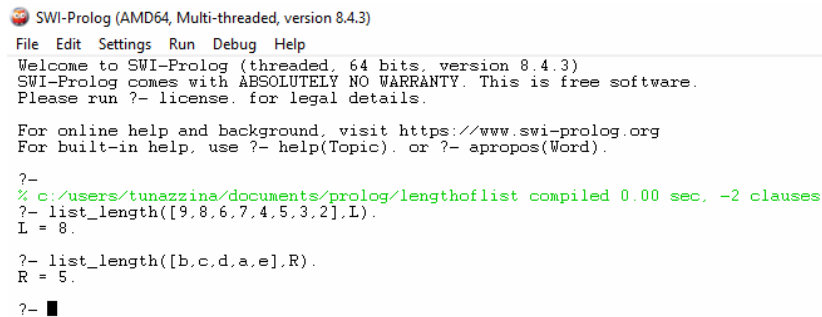
```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run ?- license. for legal details.  
  
For online help and background, visit https://www.swi-prolog.org  
For built-in help, use ?- help(Topic). or ?- apropos(Word).  
  
?-  
% c:/users/tunazzina/documents/prolog/p3 compiled 0.00 sec, -2 clauses  
?- list_member(a,[b,a,c,d]).  
true.  
  
?- list_member(a,[a,b,d]).  
true.  
  
?- list_member(a,[b,e]).  
false.  
  
?-
```

Length calculation

Program

```
list_length([],0).  
list_length([_|TAIL],N) :- list_length(TAIL,N1), N is N1 + 1.
```

Query:



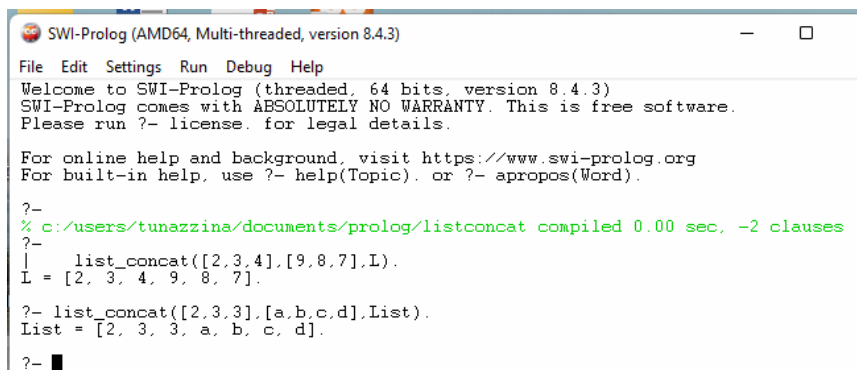
```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)  
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.  
Please run ?- license. for legal details.  
  
For online help and background, visit https://www.swi-prolog.org  
For built-in help, use ?- help(Topic). or ?- apropos(Word).  
  
?-  
% c:/users/tunazzina/documents/prolog/lengthoflist compiled 0.00 sec, -2 clauses  
?- list_length([9,8,6,7,4,5,3,2],L).  
L = 8.  
  
?- list_length([b,c,d,a,e],R).  
R = 5.  
  
?- ■
```

Concatenation of two lists

Program:

```
list_concat([],L,L).  
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
```

Query:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)  
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).  
  
?-  
% c:/users/tunazzina/documents/prolog/listconcat compiled 0.00 sec, -2 clauses  
?-  
| list_concat([2,3,4],[9,8,7],L).  
L = [2, 3, 4, 9, 8, 7].  
  
?- list_concat([2,3,3],[a,b,c,d],List).  
List = [2, 3, 3, a, b, c, d].  
  
?- ■
```

Delete from List

Program:

```
list_delete(X, [X], []).  
list_delete(X,[X|L1], L1).  
list_delete(X, [Y|L2], [Y|L1]) :- list_delete(X,L2,L1).
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help

?-
% c:/users/tunazzina/documents/prolog/delete compiled 0.00 sec, 0 clauses
?- list_delete(3,[3,4,2,1],L).
L = [4, 2, 1] .

?- list_delete(2,[9,8,4,3,2,5,6],R).
R = [9, 8, 4, 3, 5, 6] .

?-
```

Insert into List

Program

```
list_delete(X, [X], []).
list_delete(X, [X|L1], L1).
list_delete(X, [Y|L2], [Y|L1]) :- list_delete(X,L2,L1).
list_insert(X,L,R) :- list_delete(X,R,L).
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help

list_insert(5,[6,4,3,1],P).
P = [5, 6, 4, 3, 1] .

?- list_insert(p,[b,e,l,l,o],R).
R = [p, b, e, l, l, o] .

?-
```

Union of two sets

Program

```
Union([],S,S).
union(S,[],S).

union([E|T],S2,S) :-
    member(E,S2), union(T,S2,S).
union([E|T],S2,[E|S]) :-
    union(T,S2,S).
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
|
|
|      union([9,8,7,6],[3,4,2],R).
R = [9, 8, 7, 6, 3, 4, 2].
|
|      ?- union([a,b,c,d],[e,f,g],L).
L = [a, b, c, d, e, f, g].
|
|      ?-
```

Intersection of two sets

Program

```
intersect([],S,[]).

intersect([E|T],S2,[E|S]):-
    member(E,S2), intersect(T,S2,S).
intersect([E|T],S2,S):-
    intersect(T,S2,S).
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
|
|
|
|
|      intersect([a,b,c],[c,d,e],R).
R = [c]
```

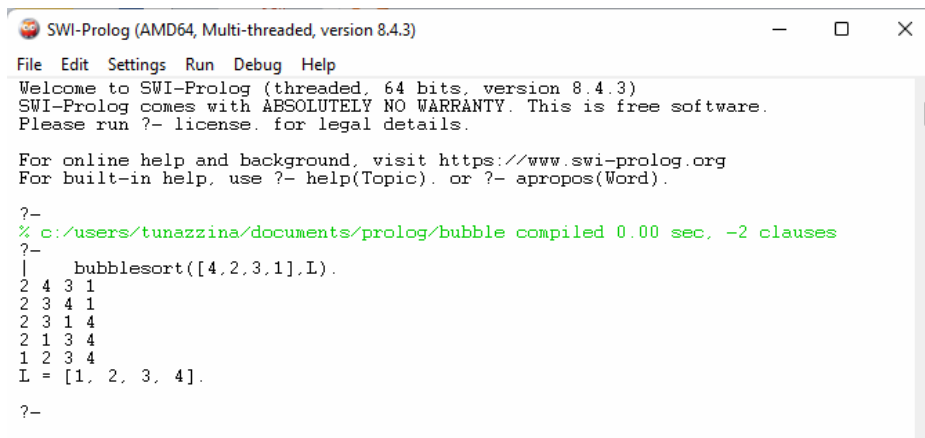

Sorting Lists : BubbleSort

Program

```
bubblesort(InputList,SortList):-
    swap(InputList,List),!,
    printlist(List),
    bubblesort(List,SortList).
bubblesort(SortList,SortList).
swap([X,Y|List],[Y,X|List]):- X > Y .
swap([Z|List],[Z|List1]):-
    swap(List,List1).

printlist([]):-nl.
printlist([Head|List]):-
    write(Head),write(' '),
    printlist(List).
```

Query:



```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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).

?-
% c:/users/tunazzina/documents/prolog/bubble compiled 0.00 sec, -2 clauses
?-
| bubblesort([4,2,3,1],L).
2 4 3 1
2 3 4 1
2 3 1 4
2 1 3 4
1 2 3 4
L = [1, 2, 3, 4].
?-
```

Conclusion: We have successfully implemented some experiment related to list .And finally we understand well how the list work in prolog programming.

Experiment No:04

Experiment Name: Prolog program on Loop, Graph, Tree, Menu driven program and DFA.

Objective:

To learn how looping is used in prolog program ,how Graph, Tree ,Menu driven program and DFA are implemented in prolog program.

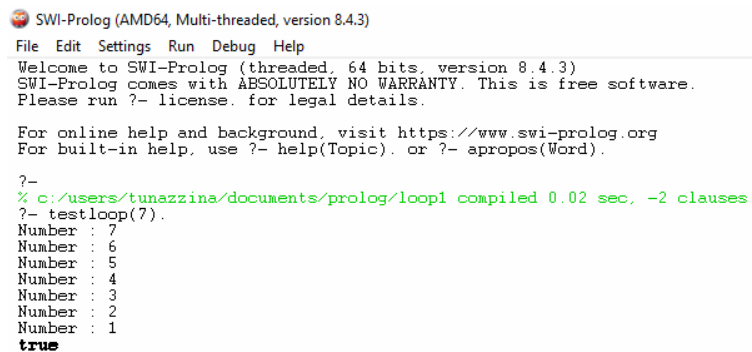
Loop in prolog

Prolog program to print numbers.

Program

```
testloop(0).  
testloop(N):- N>0, write('Number : '), write(N),  
nl, M is N-1, testloop(M).
```

Query:



The screenshot shows the SWI-Prolog (AMD64, Multi-threaded, version 8.4.3) interface. The menu bar includes File, Edit, Settings, Run, Debug, and Help. The main window displays the following text:


```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)  
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).  
  
?-  
% c:/users/tunazzina/documents/prolog/loop1 compiled 0.02 sec, -2 clauses  
?- testloop(7).  
Number : 7  
Number : 6  
Number : 5  
Number : 4  
Number : 3  
Number : 2  
Number : 1  
true
```

Menu driven program

Program

```
menu:-write("----Area calculation-----"),nl,
      write("1.Calculate area of circle"),nl,
      write("2.Calculate area of square"),nl,
      write("3.Calculate area of sphere"),nl,
      read(Choice),
      (   Choice=1,
          write("Enter radius"),read(R),
          Result is 3.14*R*R,write("Area of circle =
"),write(Result),nl,menu;
        Choice=2,
          write("Enter side of square"),read(A),
          Result is A*A,write("Area of square =
"),write(Result),nl,menu;
        Choice=3,
          write("Enter radius"),read(R),
          Result is 4*3.14*R*R,write("Area of sphere =
"),write(Result),nl,menu).
```

Query:

 SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

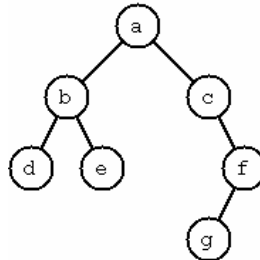
For online help and background, visit <https://www.swi-prolog.org>
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/users/tunazzina/documents/prolog/menu based compiled 0.00 sec, -2 clauses
?- menu.
----Area calculation-----
1.Calculate area of circle
2.Calculate area of square
3.Calculate area of sphere
|: 1.
Enter radius|: 5.
Area of circle = 78.5
----Area calculation-----
1.Calculate area of circle
2.Calculate area of square
3.Calculate area of sphere
|:

Prolog program for Tree

Theory:

Given tree is following



Program

```
edge(a,b) .  
edge(a,c) .  
edge(b,d) .  
edge(b,e) .  
edge(c,f) .  
edge(f,g) .
```

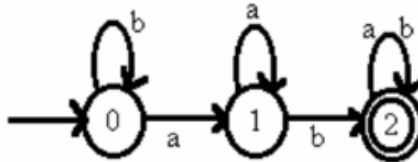
```
route(A,B) :- edge(A,C) , route(C,B) .  
route(A,B) :- edge(A,B) .
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)  
File Edit Settings Run Debug Help  
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)  
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).  
  
?-  
% c:/users/tunazzina/documents/prolog/tree compiled 0.00 sec, -2 clauses  
?- route(a,b).  
true .  
  
?-  
| route(a,d).  
true .  
  
?- route(a,f).  
true .  
  
?- route(a,g).  
true
```

Prolog Program on Graph and DFA

Theory: Given DFA is following



Program

```
start(0).
delta(0,a,1).
delta(0,b,0).
delta(1,a,1).
delta(1,b,2).
delta(2,a,2).
delta(2,b,2).
final(2).
parse(L) :- start(S), trans(S,L).
trans(S,[]):- final(S), write(S), write(' '), write([]), nl.
trans(S,[A|B]) :- delta(S,A,S1), /* S ---A---> S1 */

write(S), write(' '),
write([A|B]), nl, trans(S1,B).
```

Query:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.4.3)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.3)
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).

?-
% c:/users/tunazzina/documents/prolog/graphand dfa compiled 0.00 sec, -2 clauses
% c:/users/tunazzina/documents/prolog/graphand dfa compiled 0.00 sec, 0 clauses
% c:/users/tunazzina/documents/prolog/graphand dfa compiled 0.00 sec, 0 clauses
?- parse([b,a,a,a,a,b,a]).
0 [b,a,a,a,a,b,a]
0 [a,a,a,a,a,b,a]
1 [a,a,a,a,b,a]
1 [a,a,a,b,a]
1 [a,a,b,a]
1 [a,b,a]
1 [b,a]
2 [a]
2 []
true
```

