Name:Yash Mahajan Class:SE IT B Roll No.:04 PID:19101

St. Francis Institute of Technology
Borivli (West), Mumbai-400103
Department of Information Technology

Experiment – 12

1. Aim: To implement recursion and database manipulations using Prolog

2. Objective: After performing the experiment, the students will be able to implement

- Recursion using Prolog
- Perform simple database manipulations

3. Lab objective mapped: To understand, formulate and implement declarative programming paradigm through logic programming (PSO2) (PO1)

4. Prerequisite: knowledge of facts, rules, creation of knowledge and list comprehensions

5. Requirements: The following are the requirements – Prolog Compiler

6. Pre-Experiment Theory:

Most programming languages provide loops that allow a set of instructions to be executed repeatedly either a fixed number of times or until a given condition is met. Prolog has no looping facilities. The same effects can be obtained (that enable a sequence of events to be evaluated repeatedly) through backtracking, recursion, built in predicates or a combination of both. A recursive definition in Prolog always has at-least two parts- a first fact that acts like a stopping condition and a rule that calls itself simplified. At each level the first fact is checked. If the fact is true then the recursion ends. If not the recursion continues. A recursive rule must never call itself with the same arguments. If that happens the program will never end.

A database is a collection of information which allow organization to access, manage and update the information. Databases consist of multiple tables which are capable to include different fields. Each of these databases has different fields which might be relevant to the information stored in the database. Data manipulation provides sublanguage for databases such as SQL.

Prolog is an effective language for database queries.

7. Laboratory Exercise

A. Steps to be implemented

1. Open SWI-Prolog
2. Go to ....file-> new
3. New prolog editor will open up.
4. Write your program (collection of facts and rules)
5. Save the file in the desired as .pl file
6. Follow the steps -> Save buffer, Compile buffer and Make
7. At the prompt, first change to working directory using cd command
8. Load the required file
9. To check the outputs, fire appropriate queries at the prompt

B. Program Code

1. WAP in Prolog to display n natural numbers using recursion

```
naturalNum(0). %when argument is 0 do nothing (terminate recursion)
%loop to N
naturalNum(N):-N>0,N1 is N-1,naturalNum(N1),write(N),nl,!.
```

2. WAP in Prolog to find the sum of first n natural numbers using recursion

```
naturalSum(1,1). %when argument is 1, sum is 1
%sum of N natural numbers is the sum of first N-1 natural numbers plus N
naturalSum(N,M):-N>1,N1 is N-1,naturalSum(N1,M1),M is M1+N,!.
```

3. Create a knowledge base1. To this knowledge base, using database manipulation command- (a) append facts (b) append rules (c) delete facts (d) delete rule (e) append facts at the beginning and end of knowledge base

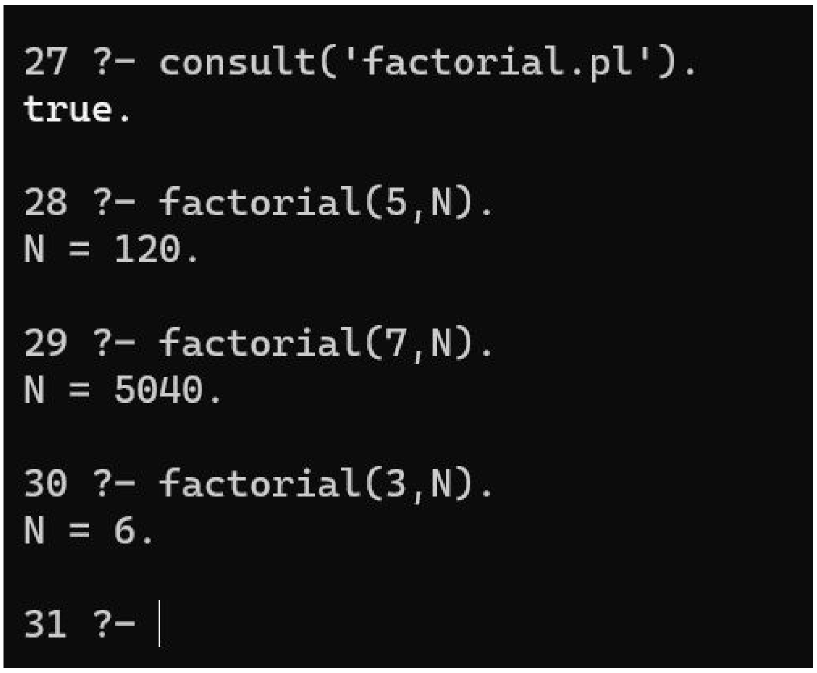
```
boy(joseph). %Joseph is a boy
boy(max).  %max is a boy
girl(tina). %Tina is a girl
girl(mia). %Mia is a girl
friends(X,Y):-girl(X),boy(Y). % X and Y are friends if X is a girl and
Y is a boy
```

8. Post Experimental Exercise

A. Questions:
1. List at least four important applications of Prolog
The applications of Prolog are as follows:-
• artificial intelligence especially in the form of Expert Systems – programs that 'reason out' the solution to complex problems using rules.
• programs for processing a 'natural language' text, to answer questions about its meaning, translate it to another language etc.
• advisory systems for legal applications
• applications for training
• maintaining databases for the Human Genome project
• a personnel system for a multinational computer company
• automatic story generation
• analysing and measuring 'social networks'
• a software engineering platform for supporting the development of complex software systems
• automatically generating legally correct licences and other documents in multiple languages
• an electronic support system for doctors.

2. WAP in Prolog to calculate factorial of a number

```
factorial(0,1). %factorial of 0 is 1
%factorial of N is factorial of N-1 times N
factorial(X,Y):- X>0,X1 is X-1,factorial(X1,Y1),Y is Y1*X,!.
```

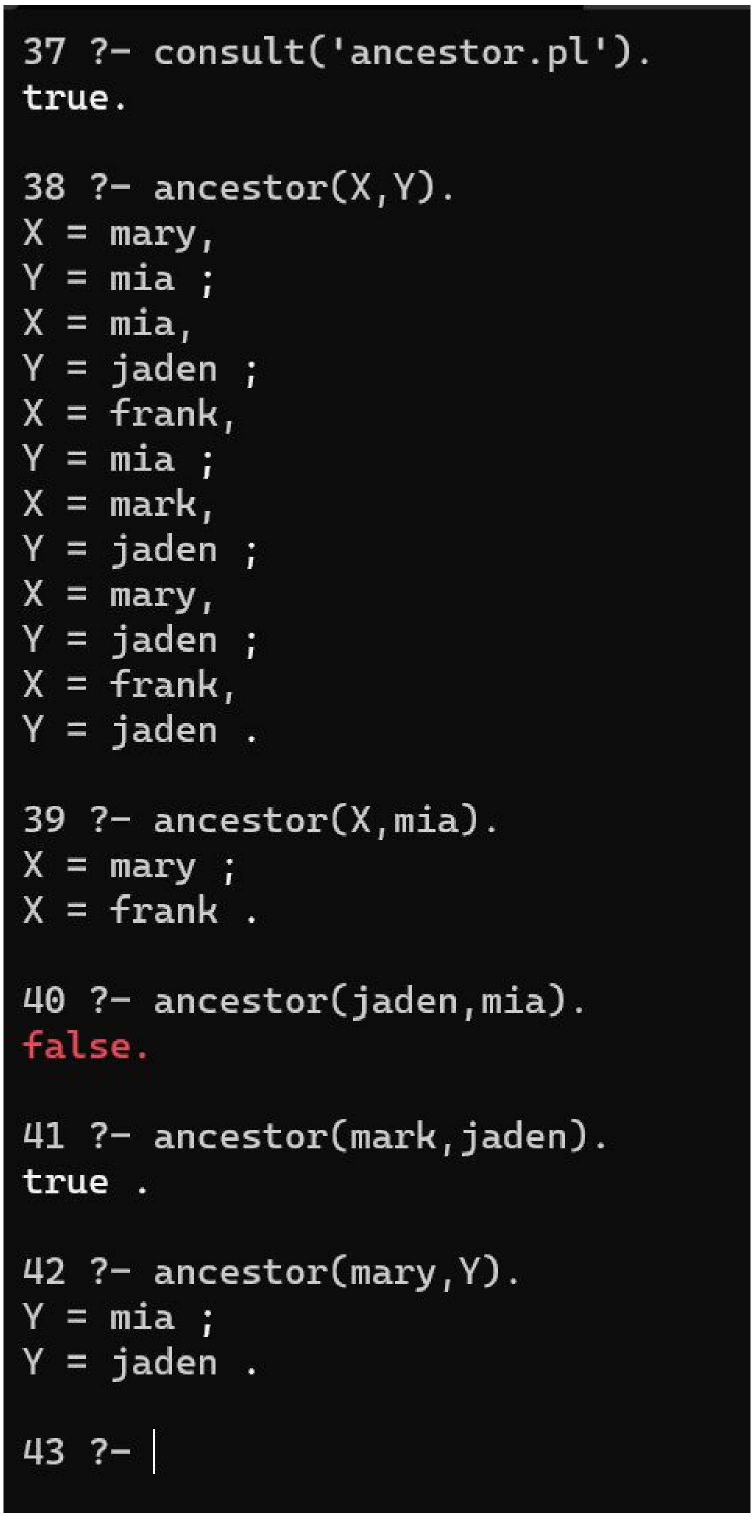


```
27 ?- consult('factorial.pl').
true.

28 ?- factorial(5,N).
N = 120.

29 ?- factorial(7,N).
N = 5040.

30 ?- factorial(3,N).
N = 6.

31 ?- 
```

From this output we can infer that the predicate factorial/2 takes 2 arguments one number and one unbound variable and return factorial of a number. The first clause in the code block signifies that factorial of 0 is a and it is the termination condition of recursion. The second line signifies that factorial of X is factorial of X-1 times X.

3. WAP in Prolog to find ancestors using database using the recursive rule

```
mother(mary,mia). %mary is mother of mia
mother(mia,jaden). %mia is mother of jaden

father(frank,mia). %frank is father of mia
father(mark,jaden). %mark is father of jaden

parent(X,Y):-mother(X,Y). %X is parent of Y  if X is mother of Y
parent(X,Y):-father(X,Y). %X is parent of Y  if X is father of Y

ancestor(X,Y):-parent(X,Y). %X is ancestor of Y if X is parent of Y
%X is ancestor of Y if X is parent of Z and Z is ancestor of Y
ancestor(X,Y):-parent(X,Z),ancestor(Z,Y).
```

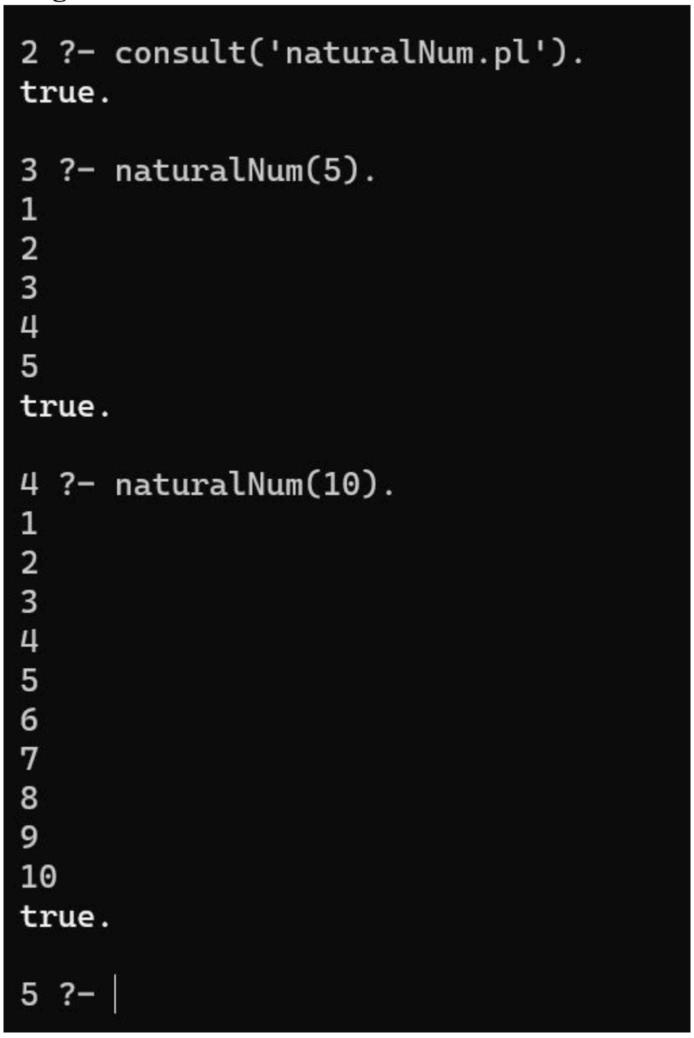


```
37 ?- consult('ancestor.pl').
true.

38 ?- ancestor(X,Y).
X = mary,
Y = mia ;
X = mia,
Y = jaden ;
X = frank,
Y = mia ;
X = mark,
Y = jaden ;
X = mary,
Y = jaden ;
X = frank,
Y = jaden .

39 ?- ancestor(X,mia).
X = mary ;
X = frank .

40 ?- ancestor(jaden,mia).
false.

41 ?- ancestor(mark,jaden).
true .

42 ?- ancestor(mary,Y).
Y = mia ;
Y = jaden .

43 ?-
```

From this output we can infer that the predicate ancestor/2 takes 2 arguments. In the code block we can see that the first clause of ancestor is that X is an ancestor of Y if X is a parent of Y. In the second line we can see that X is an ancestor of Y if X is a parent of Z and Z is an ancestor of Y. Thus we can find the ancestor of anyone using recursion.

B. Results/Observations/Program output:

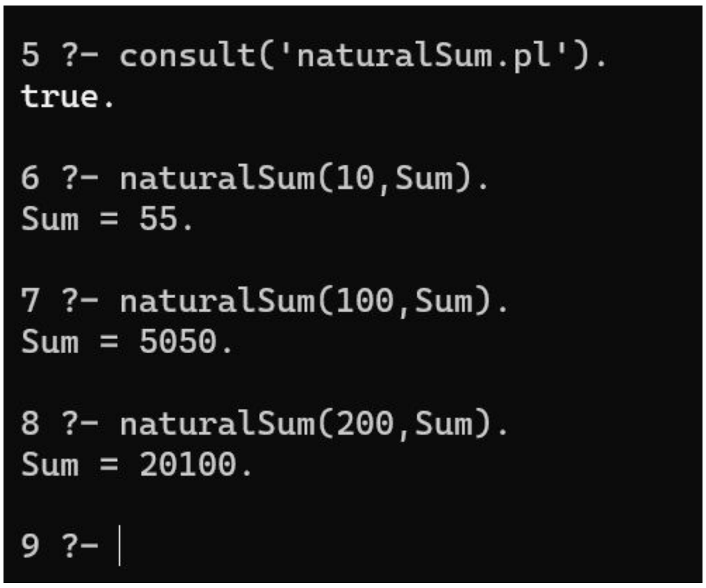
Program 1:-



```
2 ?- consult('naturalNum.pl').
true.

3 ?- naturalNum(5).
1
2
3
4
5
true.

4 ?- naturalNum(10).
1
2
3
4
5
6
7
8
9
10
true.

5 ?- |
```

From this experiment we can infer that the predicate naturalNum/1 takes one argument and displays the numbers from 1 to the number N. In the code block the first clause states that when naturalNum is 0 do nothing i.e. terminate recursion. The second clause states that N is a natural number if N is greater than 1 and N-1 is a natural number.

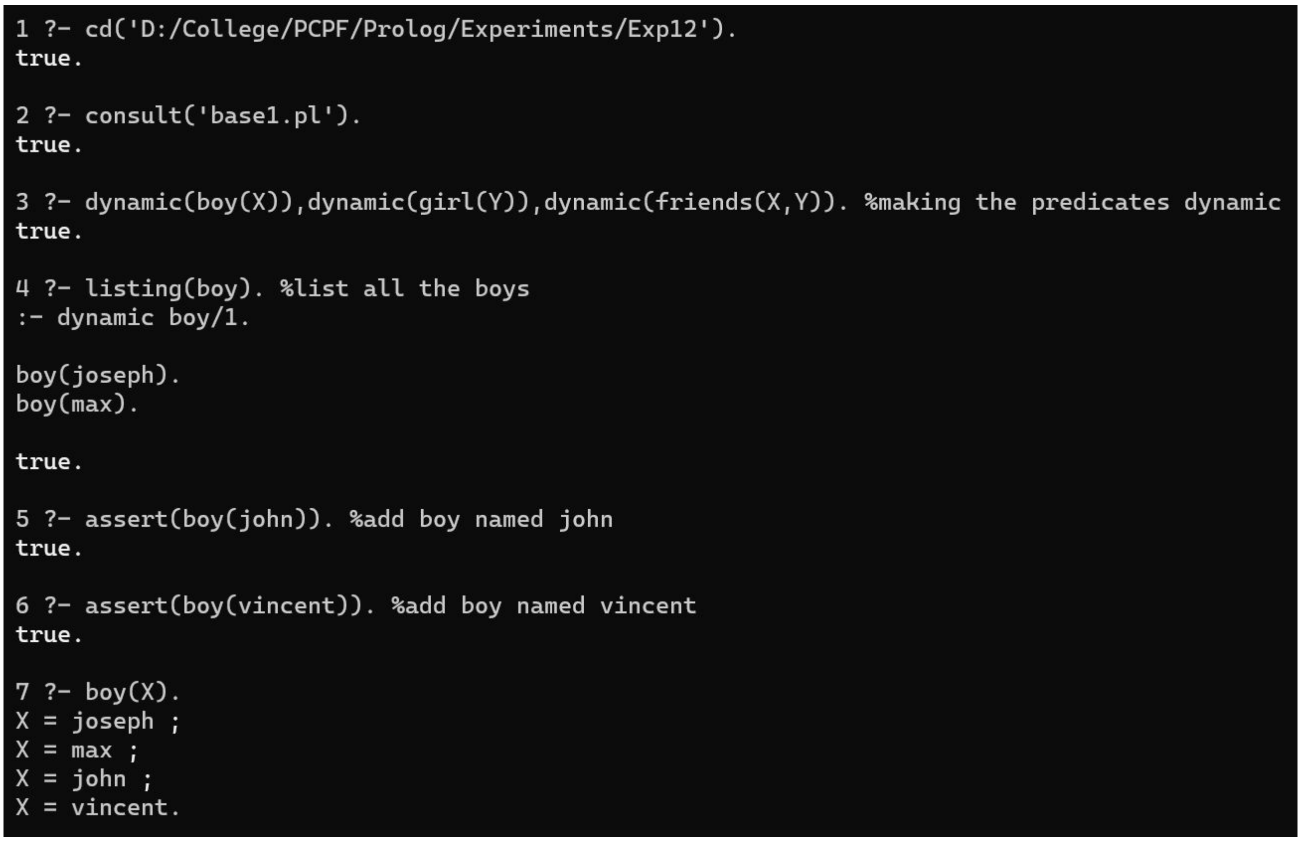
Program 2:-



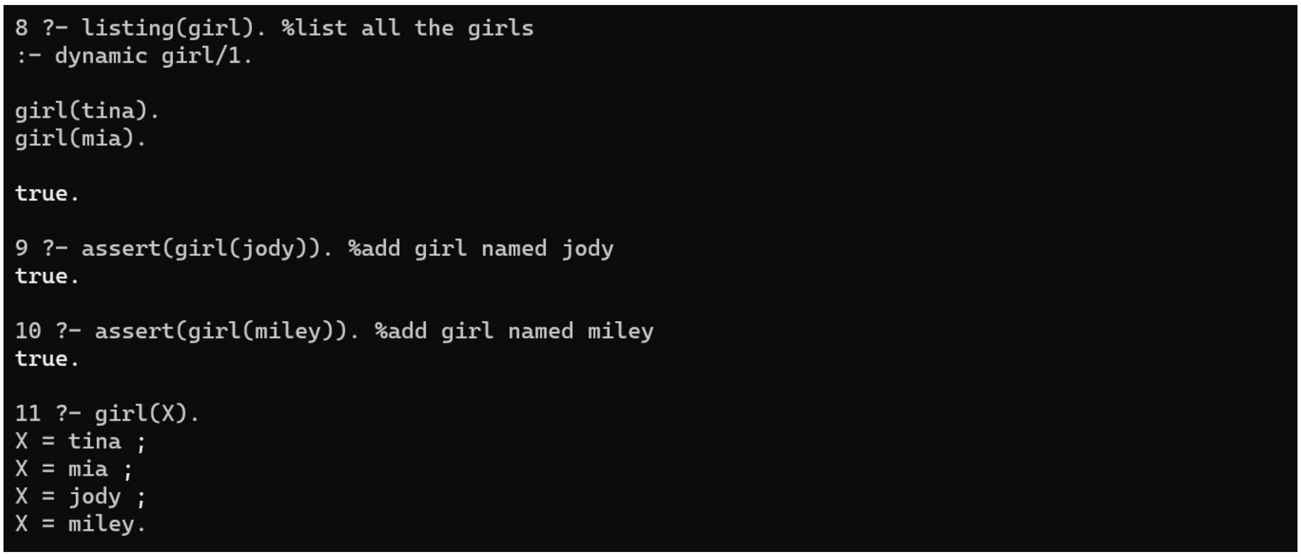
```
5 ?- consult('naturalSum.pl').
true.

6 ?- naturalSum(10,Sum).
Sum = 55.

7 ?- naturalSum(100,Sum).
Sum = 5050.

8 ?- naturalSum(200,Sum).
Sum = 20100.

9 ?- |
```

From this experiment we can infer that the predicate naturalSum/2 takes two arguments one is an integer number and second is an unbounded variable and displays the sum of numbers from 1 to the number N. In the code block the first clause states that when number is 1 is naturalSum is 1. The second clause states that the sum of N natural numbers is the sum of first N-1 natural numbers plus N.

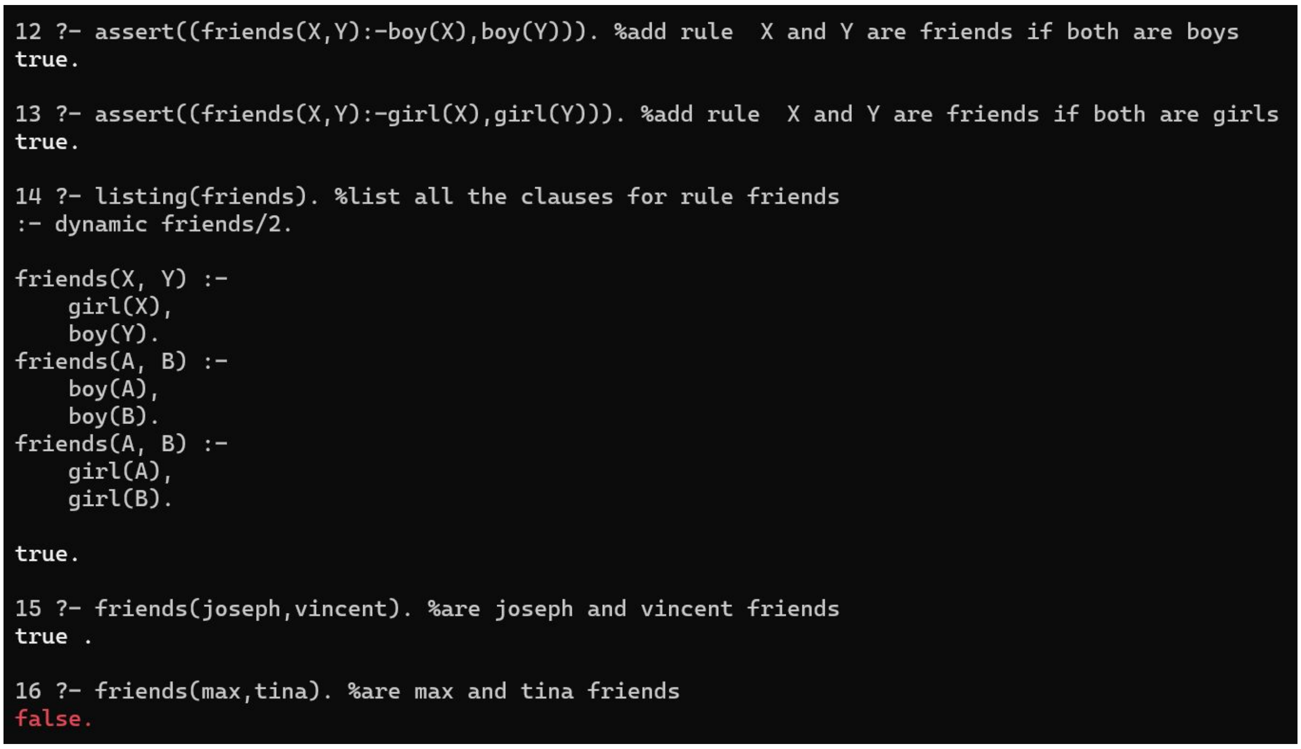
Program 3:-



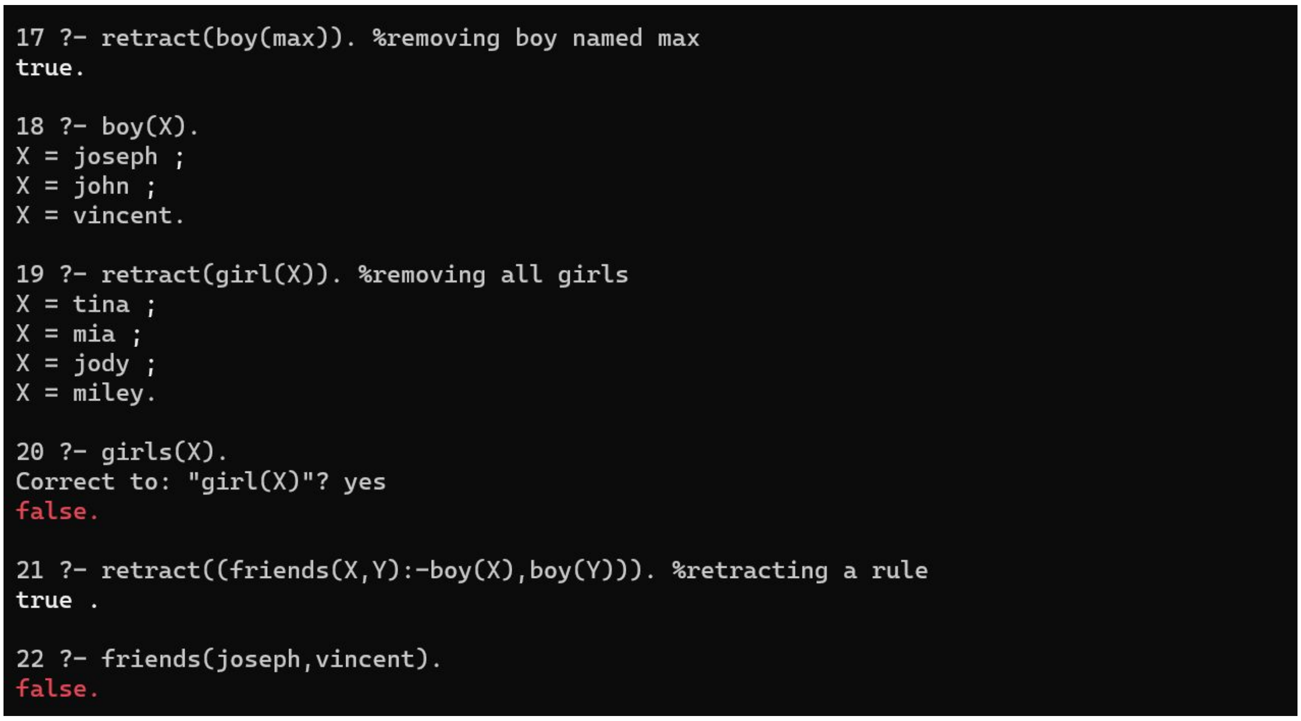
```
1 ?- cd('D:/College/PCPF/Prolog/Experiments/Exp12').
true.

2 ?- consult('base1.pl').
true.

3 ?- dynamic(boy(X)),dynamic(girl(Y)),dynamic(friends(X,Y)). %making the predicates dynamic
true.

4 ?- listing(boy). %list all the boys
:- dynamic boy/1.

boy(joseph).
boy(max).

true.

5 ?- assert(boy(john)). %add boy named john
true.

6 ?- assert(boy(vincent)). %add boy named vincent
true.

7 ?- boy(X).
X = joseph ;
X = max ;
X = john ;
X = vincent.
```



```
8 ?- listing(girl). %list all the girls
:- dynamic girl/1.

girl(tina).
girl(mia).

true.

9 ?- assert(girl(jody)). %add girl named jody
true.

10 ?- assert(girl(miley)). %add girl named miley
true.

11 ?- girl(X).
X = tina ;
X = mia ;
X = jody ;
X = miley.
```



```
12 ?- assert((friends(X,Y):-boy(X),boy(Y))). %add rule  X and Y are friends if both are boys
true.

13 ?- assert((friends(X,Y):-girl(X),girl(Y))). %add rule  X and Y are friends if both are girls
true.

14 ?- listing(friends). %list all the clauses for rule friends
:- dynamic friends/2.

friends(X, Y) :-
    girl(X),
    boy(Y).
friends(A, B) :-
    boy(A),
    boy(B).
friends(A, B) :-
    girl(A),
    girl(B).

true.

15 ?- friends(joseph,vincent). %are joseph and vincent friends
true .

16 ?- friends(max,tina). %are max and tina friends
false.
```



```
17 ?- retract(boy(max)). %removing boy named max
true.

18 ?- boy(X).
X = joseph ;
X = john ;
X = vincent.

19 ?- retract(girl(X)). %removing all girls
X = tina ;
X = mia ;
X = jody ;
X = miley.

20 ?- girls(X).
Correct to: "girl(X)"? yes
false.

21 ?- retract((friends(X,Y):-boy(X),boy(Y))). %retracting a rule
true .

22 ?- friends(joseph,vincent).
false.
```

From this output we can infer that before modifying any clauses in the knowledge base we need to specify them as dynamic , example dynamic(mypred/3). After that we can modify the knowledge base. To add more facts and rules to the existing knowledge base we use the assert predicate. Example assert(boy(tom)). To retract or remove any fact or clause we use the retract predicate. Example retract(boy(tom)).

C. Conclusion:

In this experiment we have studied recursion and database manipulations using Prolog. We have also implemented programs to study how recursion and database manipulation is carried out in prolog. We have used predicates that cause a sequence of goals to be evaluated repeatedly, either a fixed number of times or until a specified condition is satisfied and predicates which causes one or more clauses to be added to or deleted from the Prolog database.

To perform this experiment we have used the SWI-Prolog 8.2.1 console and editor.

From this experiment we infer that most conventional programming languages have a looping facility that enables a set of instructions to be executed repeatedly either a fixed number of times or until a given condition is met. Prolog has no looping facilities, similar effects can be obtained that enable a sequence of goals to be evaluated repeatedly. This can be done in a variety of ways, using backtracking, recursion, built-in predicates, or a combination of these. Prolog also has built-in predicates for adding clauses to and deleting clauses from the database which can be useful for more advanced programming in the language.

D. References:

[1] Michael L Scott, "Programming Language Pragmatics", Third edition, Elsevier publication
[2] Max Bramer, "Logic Programming with Prolog", Springer, 2005
[3]https://boklm.eu/prolog/page_6.html#:~:text=In%20Prolog%2C%20recursion%20appears%20when,another%20rule%20or%20call %20itself.