A
LAB REPORT OF
ARTIFICIAL INTELLIGENCE

Submitted By
Sabina Bogati
Roll No: 12095/20

Submitted to:
Mr. Dinesh Maharjan

Department of Management
Nepal Commerce Campus
Tribhuvan University

In the partial fulfilment of the requirement for the course Artificial Intelligence.

Min Bhawan, Kathmandu
December, 2024

Index

LAB I: WHAT IS PROLOG? DESCRIBE ITS HISTORY AND CURRENT USES IN BREIF

Objectives:

The objectives of the lab dealt with understanding and exploring Prolog, a logic programming language commonly used in artificial intelligence and computational linguistics.

Theory:

Prolog is a declarative programming language designed for developing logic-based AI applications. We can set rules and facts around a problem, and then Prolog's interpreter will use that information to automatically infer solutions.

It's a declarative programming language, meaning that it allows us to specify the rules and facts about a problem domain, and then the Prolog interpreter will use these rules and facts to automatically infer solutions to problems.

One of the key features of Prolog is its ability to handle uncertain or incomplete information. In Prolog, we can specify a set of rules and facts that are known to be true, but they can also specify rules and facts that might be true or false.

To use Prolog, we will need to have a Prolog interpreter installed on your computer. There are several different Prolog interpreters available, including SWI-Prolog, GNU Prolog and B-Prolog.

Facts in prolog:

In Prolog, facts are statements that are assumed to be true. They are used to provide the interpreter with information about the problem domain, and the interpreter will use this information to automatically infer solutions to problems.

For example,

```
animal(dog).
woman(sabina).
capital_of(Japan,  Tokyo).
```

In this example, the first line states that dog is an animal, the second line states that Sabina is a woman, and the third line states that Tokyo is the capital of Japan.

Variables in prolog:

In Prolog, variables are used to represent values that can change or be determined by the interpreter. They are written using a name that begins with an uppercase letter, such as X or Y.

For example,

```
capital_of(Country, Capital).
```

In this case, the fact states that the Capital of a given Country is unknown, and the interpreter will use other facts and rules to determine the value of Capital when a query is made.

Queries in prolog:

Queries are used to ask the interpreter to find solutions to problems based on the rules and facts in the program. In Prolog, queries are written using the same syntax as facts, followed by a question mark (?).

For example,

```
capital_of(Japan, X)?
```

In this query, the interpreter will use the capital_of/2 fact that was defined earlier to determine that the capital of Japan is Tokyo, and it will return the value Tokyo for the variable X as the solution.

History of prolog:

The logic programming language PROLOG (Programmation en Logique) was conceived by Alain Colmerauer at the University of Aix-Marseille, France, where the language was first implemented in 1973. PROLOG was further developed by the logician Robert Kowalski, a member of the AI group at the University of Edinburgh.

Prolog has been used largely for logic programming, and its applications include natural language understanding and expert systems such as MYCIN.

To this day Prolog has grown in use throughout North America and Europe. Prolog was used heavily in the European Esprit programme and in Japan where it was used in building the ICOT Fifth Generation Computer Systems Initiative.

Use cases of prolog:

Logic programming can be used in any domain where a large amount of data must be analyzed to make decisions. However, it is most commonly applied to a few subjects.

- Artificial Intelligence/Machine Learning: This is one of the main applications of logic programming. It is especially relevant because it provides a structured method of defining domain-specific knowledge. AI systems use their facts and rules to analyze new queries and statements.
- Natural Language Processing (NLP): NLP handles interactions between people and computers. It relies upon a system of rules to interpret and understand speech or text. NLP systems translate their insights back into a more data-friendly format. NLP systems can also generate a relevant response to user requests and feedback.
- Database Management: Logic programming can determine the best place in a database to store new data. It can also analyze the contents of a database and retrieve the most useful and relevant results for a query. Logic programming is frequently used with large freeform NoSQL databases. These databases do not use tables to organize and structure data and must be analyzed using other methods.
- Predictive Analysis: Logic programs can sort through a large amount of data, analyze results and make predictions. This is especially useful in areas such as climate forecasting, the monitoring of deep space objects, and predicting equipment failures.

Observation:

Prolog's simplified syntax simplifies logic-based programming, enabling intuitive definition of relationships through predicates and rules. Exploring its declarative nature provided insights into a distinct programming paradigm, while its historical context highlighted its relevance across diverse fields.

Conclusion:

This lab deepened understanding of Prolog's principles and applications, fostering proficiency in problem-solving and appreciation for declarative programming. Moving forward, opportunities to explore advanced features and tackle complex domains underscore its value in both academic and practical settings.

LAB II : WRITE A PROGRAM TO DISPLAY YOUR NAME IN PROLOG

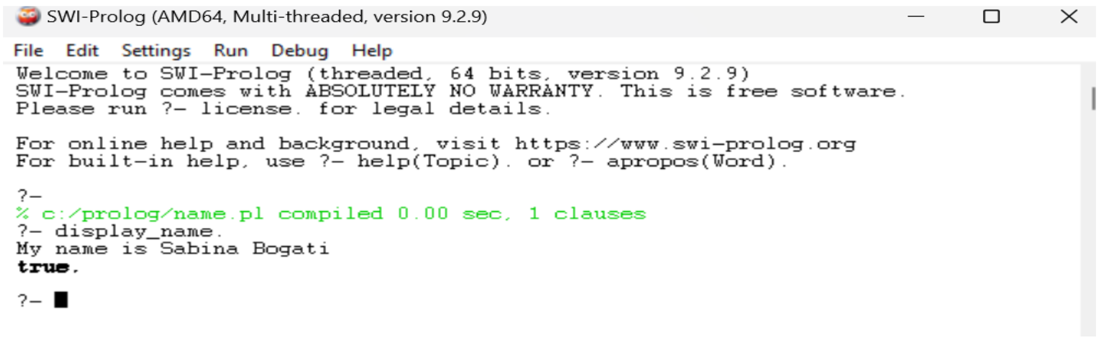
Objectives:

In this lab, I aim to understand the basics of Prolog syntax and semantics while creating a program to display my name. By accomplishing this task, I hope to gain practical experience in Prolog programming and build a foundation for more complex logic-based projects.

Source code:

```
display_name :-
    write('My name is Sabina Bogati').
```

Output:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
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:/prolog/name.pl compiled 0.00 sec, 1 clauses
?- display_name.
My name is Sabina Bogati
true.

?- 
```

Observation:

While working on the "display_name" Prolog program, I observed that the syntax for string manipulation in Prolog is straightforward and easy to understand. Using the "write" predicate to display text made the code simple and intuitive. Nonetheless, this lab served as a clear introduction to basic Prolog programming concepts, providing a solid foundation for future tasks involving logic-based programming.

Conclusion:

From the above lab I gained hands on experience in Prolog's syntax for string manipulation. Utilizing the "write" predicate to display text made the code easy to comprehend and execute. However, I also noticed the limited string manipulation capabilities inherent in Prolog compared to more modern programming languages.

LAB III : WRITE A PROLOG PROGRAM TO REPRESENT FEW BASIC FACTS AND PERFORM QUERIES

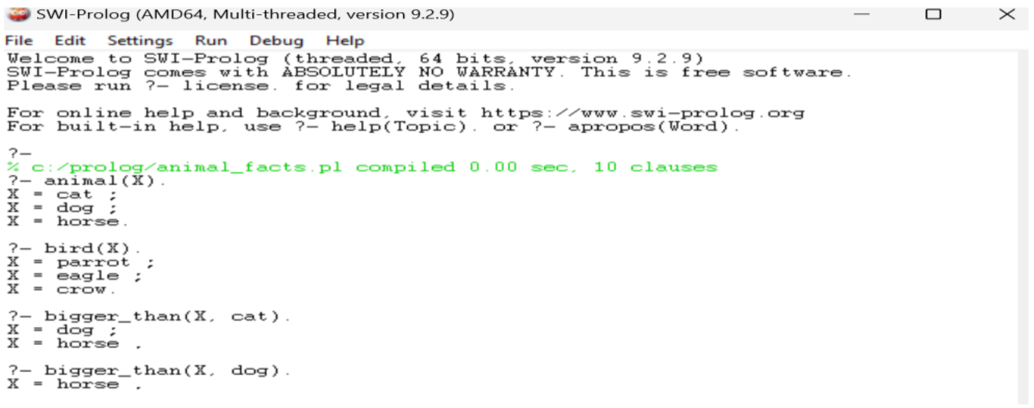
Objectives:

In this lab, my goal is to learn the fundamentals of Prolog by writing a program to represent basic facts and execute queries. By accomplishing this task, I aim to gain hands-on experience with Prolog syntax, predicates, and rules, as well as understand how to use queries to retrieve information from the knowledge base.

Source code:

```
animal(cat).
animal(dog).
animal(horse).
bird(parrot).
bird(eagle).
bird(crow).
bigger_than(dog,cat).
bigger_than(horse,dog).
bigger_than(Y,X):-bigger_than(Z,X),bigger_than(Y,Z).
bigger_than(Y,X):-bigger_than(Y,Z),bigger_than(Z,X).
```

Output:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
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:/prolog/animal_facts.pl compiled 0.00 sec, 10 clauses
?- animal(X).
X = cat ;
X = dog ;
X = horse.

?- bird(X).
X = parrot ;
X = eagle ;
X = crow.

?- bigger_than(X, cat).
X = dog ;
X = horse .

?- bigger_than(X, dog).
X = horse .
```

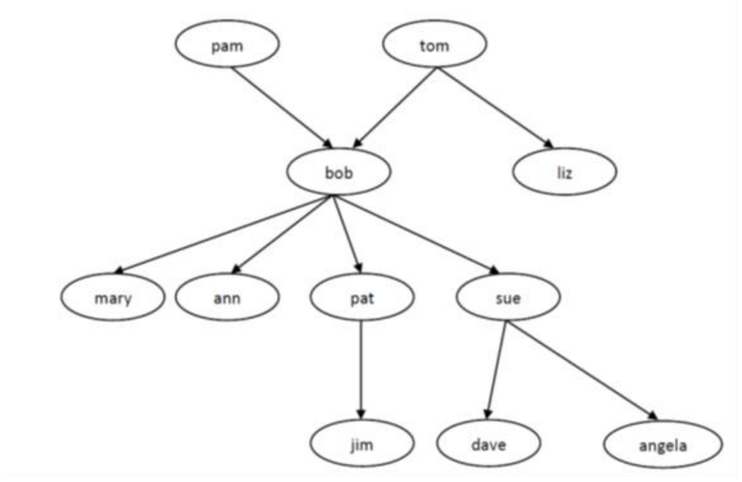
<u>Observation:</u>

During this lab work, I observed the straightforward representation of facts using simple predicates like animal/1 and bigger_than/2. Additionally, the recursive definition of the bigger_than/2 predicate demonstrated Prolog's ability to express transitive relationships between entities, allowing for concise and expressive code.

<u>Conclusion:</u>

After completing this lab, I gained hands-on experience on Prolog's declarative nature and recursive capabilities. By representing facts and defining rules, I gained a deeper understanding of how Prolog can model real-world relationships and solve logic-based problems.

LAB IV: EXPLORING FAMILY TREE IN PROLOG – I



Objectives:

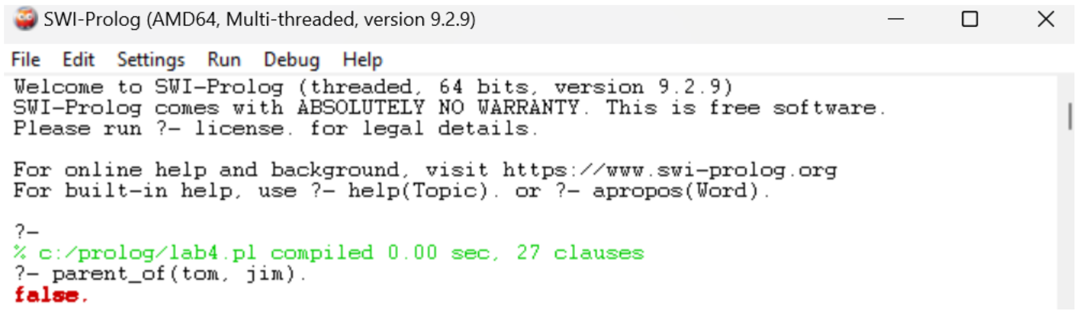
In this lab, our goal is to gain practical experience in representing family relationships using Prolog.

Source Code:

```
male(tom).
male(bob).
male(jim).
male(dave).
male(pat).

female(pam).
female(mary).
female(ann).
female(liz).
female(sue).
female(angela).

parent_of(pam, bob).
parent_of(tom, bob).
parent_of(tom, liz).
parent_of(bob, mary).
parent_of(bob, ann).
parent_of(bob, pat).
parent_of(bob, sue).
parent_of(pat, jim).
parent_of(sue, dave).
parent_of(sue, angela).

grand_parentof(P,R):-parent_of(P,Q),parent_of(Q,R).

sibling(P,R):-parent_of(X,P),parent_of(X,R),P\=R.

grandchild_of(X, Y) :-
    parent_of(Y, Z),
    parent_of(Z, X),
      (female(X) ; male(X) ).

common_parent(X, Y) :-
    parent_of(Z, X),
    parent_of(Z, Y),
    X\= Y.

sister_of(X, Y) :-
    parent_of(Z, X),
    parent_of(Z, Y),
    female(X),
    X \= Y.

uncle_of(X, Y) :-
    parent_of(Z, Y),
    sibling(X, Z),
    male(X).
```

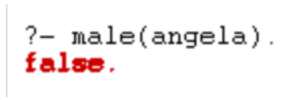
Output:

i. Is tom parent of jim?


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help

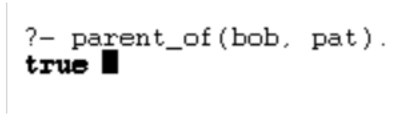
```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?-
% c:/prolog/lab4.pl compiled 0.00 sec, 27 clauses
?- parent_of(tom, jim).
false.
```

ii. Is angela male?



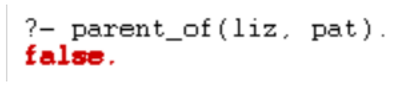
```
?- male(angela).
false.
```

iii. Is bob parent of pat?



```
?- parent_of(bob, pat).
true █
```

iv. Is liz parent of pat?



```
?- parent_of(liz, pat).
false.
```

<u>Observation:</u>

During this lab, it became evident that representing family members and their relationships in Prolog was a straightforward task. Utilizing properties to denote gender and establishing parent-child connections via factual declarations facilitated a clear depiction of the family structure.

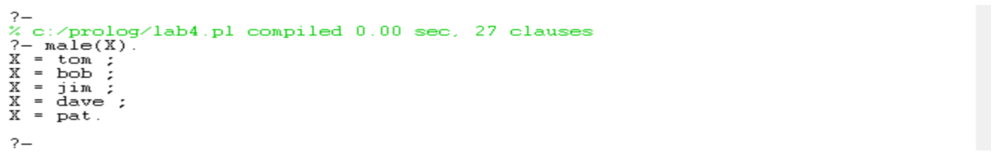
<u>Conclusion:</u>

After completion of this lab, I gained hands-on experience in organizing and interrogating familial relationships. By utilizing Prolog's capabilities to define familial attributes and infer connections, a deeper comprehension of relational reasoning was attained.

LAB V: EXPLORING FAMILY TREE IN PROLOG – II

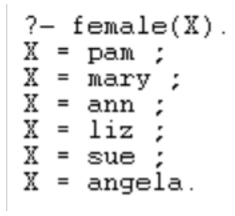
<u>Output:</u>

i. List all the male family members.



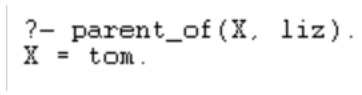
```
?-
% c:/prolog/lab4.pl compiled 0.00 sec, 27 clauses
?- male(X).
X = tom ;
X = bob ;
X = jim ;
X = dave ;
X = pat.

?-
```

ii. List all female family members.



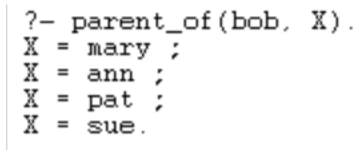
```
?- female(X).
X = pam ;
X = mary ;
X = ann ;
X = liz ;
X = sue ;
X = angela.
```

iii. Who is liz's parent?



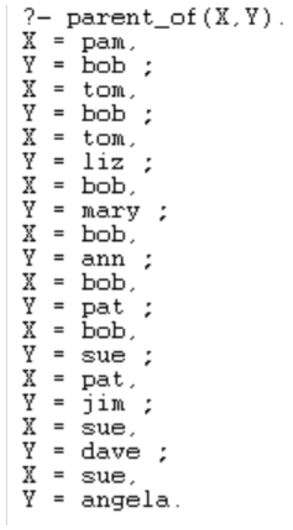
```
?- parent_of(X, liz).
X = tom.
```

iv. Who are bob children?



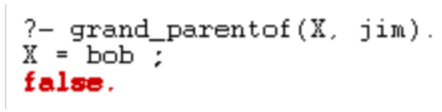
```
?- parent_of(bob, X).
X = mary ;
X = ann ;
X = pat ;
X = sue.
```

v. Find all parent children relationship.



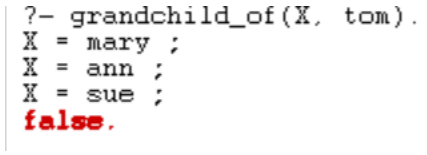
```
?- parent_of(X,Y).
X = pam,
Y = bob ;
X = tom,
Y = bob ;
X = tom,
Y = liz ;
X = bob,
Y = mary ;
X = bob,
Y = ann ;
X = bob,
Y = pat ;
X = bob,
Y = sue ;
X = pat,
Y = jim ;
X = sue,
Y = dave ;
X = sue,
Y = angela.
```

vi. Who is grand parent of jim?



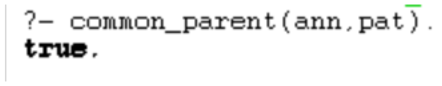
```
?- grand_parentof(X, jim).
X = bob ;
false.
```

vii. Who are tom’s granddaughters?



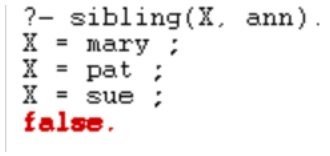
```
?- grandchild_of(X, tom).
X = mary ;
X = ann ;
X = sue ;
false.
```

viii. Does ann and pat have common parent?



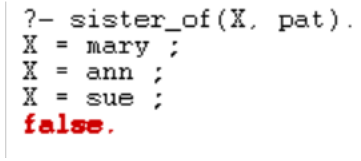
```
?- common_parent(ann,pat).
true.
```

ix. Who are siblings of ann?



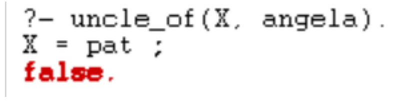
```
?- sibling(X, ann).
X = mary ;
X = pat ;
X = sue ;
false.
```

x. List all the sisters of pat.



```
?- sister_of(X, pat).
X = mary ;
X = ann ;
X = sue ;
false.
```

xi. List all the uncles of Angela.



```
?- uncle_of(X, angela).
X = pat ;
false.
```

LAB VI: WRITE A PROGRAM TO DISPLAY FACTORIAL OF A USER DEFINED NUMBER USING RECURSION

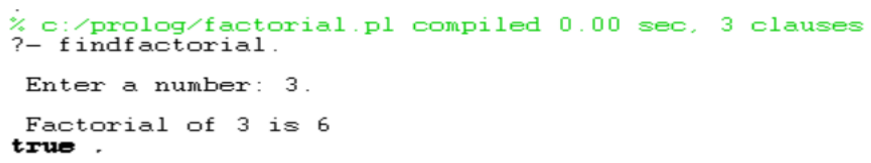
Objectives:

The objective of this lab is to create a program in Prolog that can calculate the factorial of any number provided by the user by using recursion.

Source Code:

```
factorial(0,1).
factorial(N,F):-
     N>0,
     N1 is N-1,
     factorial(N1,F1),
     F is N*F1.
findfactorial:-
     write('\n Enter a number: '),
     read(Num),
     factorial(Num,F),
     write('\n Factorial of '),write(Num),write(' is '),write(F).
```

Output:



```
% c:/prolog/factorial.pl compiled 0.00 sec, 3 clauses
?- findfactorial.

 Enter a number: 3.

 Factorial of 3 is 6
true .
```

Observation:

During this lab, I observed the simplicity of Prolog's recursive approach, breaking down the factorial problem into smaller steps until reaching the base case of 0. Additionally, using the read and write predicates allowed for interactive user input and output.

Conclusion:

After completion of this lab, I gained hands on experience on using recursion and arithmetic input/output in prolog.

LAB VII: WRITE A PROGRAM TO DISPLAY FIBONACCI SEQUENCE UP TO USER DEFINED NUMBER.

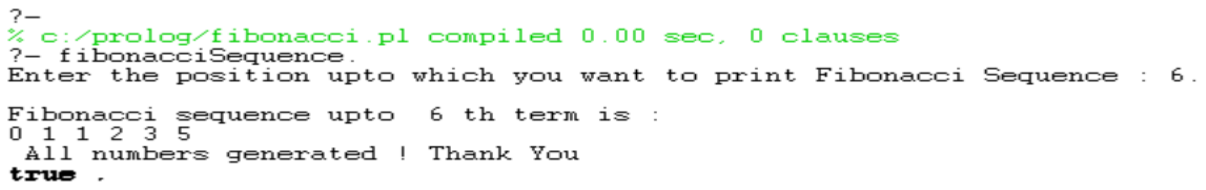
Objectives:

The objective of this lab is to create a program in Prolog that can calculate the fibonacci sequence up to any number provided by the user by using recursion.

Source code:

```
fibonacciSequence:-
write('Enter the position upto which you want to print Fibonacci
Sequence : '),read(N),nl,
    write('Fibonacci sequence upto  '),write(N),write(' th term is
: '),nl,
    A is 0,
    B is 1,
    write(A),write(' '),write(B),write(' '),
      Num is N-1,
    fibonacci(Num,A,B).

fibonacci(N,A,B):-
        N<2, write(' \n All numbers generated ! Thank You');
        C is A+B,
        write(C),write(' '),
        D is B,
        E is C,
        N1 is N-1,
        fibonacci(N1,D,E).
```

Output:



```
?-
% c:/prolog/fibonacci.pl compiled 0.00 sec, 0 clauses
?- fibonacciSequence.
Enter the position upto which you want to print Fibonacci Sequence : 6.

Fibonacci sequence upto  6 th term is :
0 1 1 2 3 5
 All numbers generated ! Thank You
true .
```

LAB VIII: WRITE A PROGRAM TO IMPLEMENT MONKEY BANANA PROBLEM

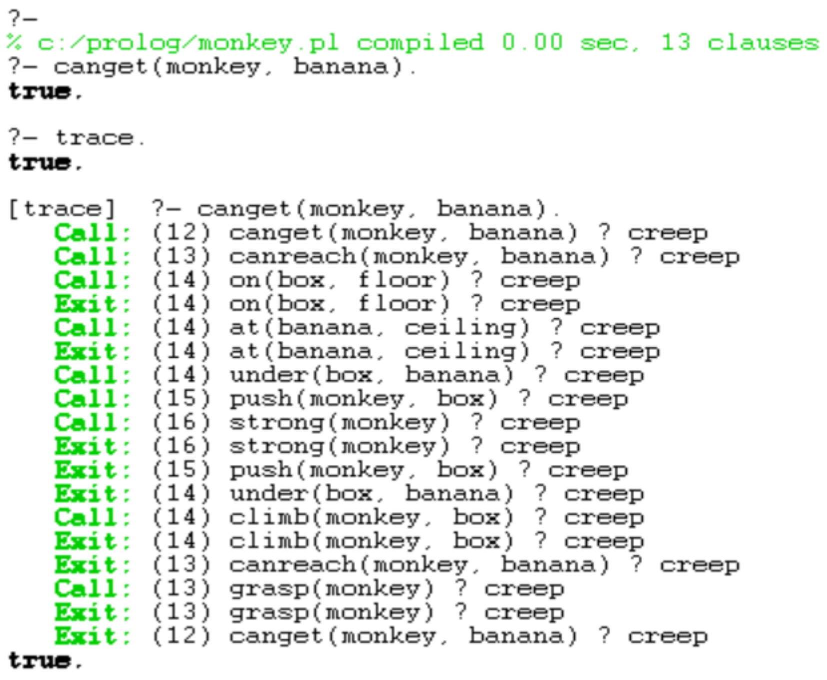
Objectives:

The aim of this lab is to create a Prolog program that tackles the Monkey Banana Problem, helping a monkey get a banana while dealing with hurdles.

Source code:

```
on(monkey,floor).
on(box,floor).
in(monkey,room).
in(box,room).
in(banana,room).
at(banana,ceiling).

strong(monkey).
grasp(monkey).
climb(monkey,box).

push(monkey,box):-
     strong(monkey).
under(box,banana):-
     push(monkey,box).

canreach(monkey,banana):-
     on(box,floor),
     at(banana,ceiling),
     under(box,banana),
     climb(monkey,box).

canget(monkey,banana):-
     canreach(monkey,banana),
     grasp(monkey).
```

Output:



```
?-
% c:/prolog/monkey.pl compiled 0.00 sec, 13 clauses
?- canget(monkey, banana).
true.

?- trace.
true.

[trace]  ?- canget(monkey, banana).
   Call: (12) canget(monkey, banana) ? creep
   Call: (13) canreach(monkey, banana) ? creep
   Call: (14) on(box, floor) ? creep
   Exit: (14) on(box, floor) ? creep
   Call: (14) at(banana, ceiling) ? creep
   Exit: (14) at(banana, ceiling) ? creep
   Call: (14) under(box, banana) ? creep
   Call: (15) push(monkey, box) ? creep
   Call: (16) strong(monkey) ? creep
   Exit: (16) strong(monkey) ? creep
   Exit: (15) push(monkey, box) ? creep
   Exit: (14) under(box, banana) ? creep
   Call: (14) climb(monkey, box) ? creep
   Exit: (14) climb(monkey, box) ? creep
   Exit: (13) canreach(monkey, banana) ? creep
   Call: (13) grasp(monkey) ? creep
   Exit: (13) grasp(monkey) ? creep
   Exit: (12) canget(monkey, banana) ? creep
true.
```

LAB IX: WRITE A PROGRAM TO IMPLEMENT TOWER OF HANOI PROBLEM.

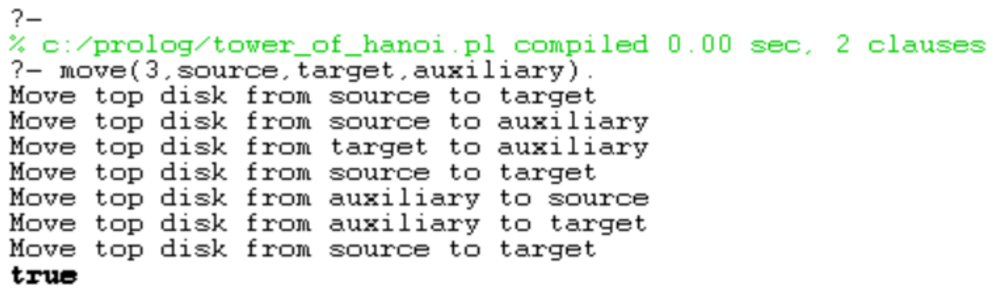
Objectives:

Develop a Prolog program to solve the Tower of Hanoi problem, aiming to move a stack of disks from one peg to another, obeying the rules of the puzzle.

Source code:

```
move(1,X,Y,_) :-

   write('Move top disk from '), write(X), write(' to '), write(Y),
nl.

move(N,X,Y,Z) :-

   N>1,

   M is N-1,

   move(M,X,Z,Y),

   move(1,X,Y,_),

   move(M,Z,Y,X).
```

Output:



```
?-
% c:/prolog/tower_of_hanoi.pl compiled 0.00 sec, 2 clauses
?- move(3,source,target,auxiliary).
Move top disk from source to target
Move top disk from source to auxiliary
Move top disk from target to auxiliary
Move top disk from source to target
Move top disk from auxiliary to source
Move top disk from auxiliary to target
Move top disk from source to target
true
```

LAB X: FACT REPRESENTATION USING PROLOG

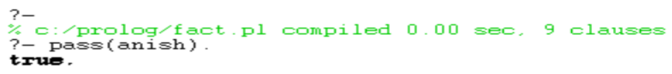
- If a student study they will pass the exam.
- If the student passes their exam, they will be happy.
- Radha studied for the exam.
- Rakesh studied for the exam.
- Anish studied for the exam.
- Rekha did not study for her exam.
- Bibek did not study for the exam

Source Code:

```
pass(X):-study(X).
happy(X):-pass(X).
notstudy(rekha).
notstudy(bibek).
study(radha).
study(rakesh).
study(anish).
fail(X):-notstudy(X).
unhappy(X):-fail(X).
```

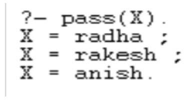
Output:

i. Did Anish pass?



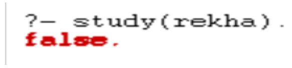
```
?-
% c:/prolog/fact.pl compiled 0.00 sec, 9 clauses
?- pass(anish).
true.
```

ii. List all the students that pass.



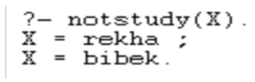
```
?- pass(X).
X = radha ;
X = rakesh ;
X = anish.
```

iii. Did Rekha study?



```
?- study(rekha).
false.
```

iv. List all the students that did not study?



```
?- notstudy(X).
X = rekha ;
X = bibek.
```

v.

LAB XI: WRITE A PROGRAM USING C OR JAVA TO IMPLEMENT AND, OR, NOT AND XOR GATE.

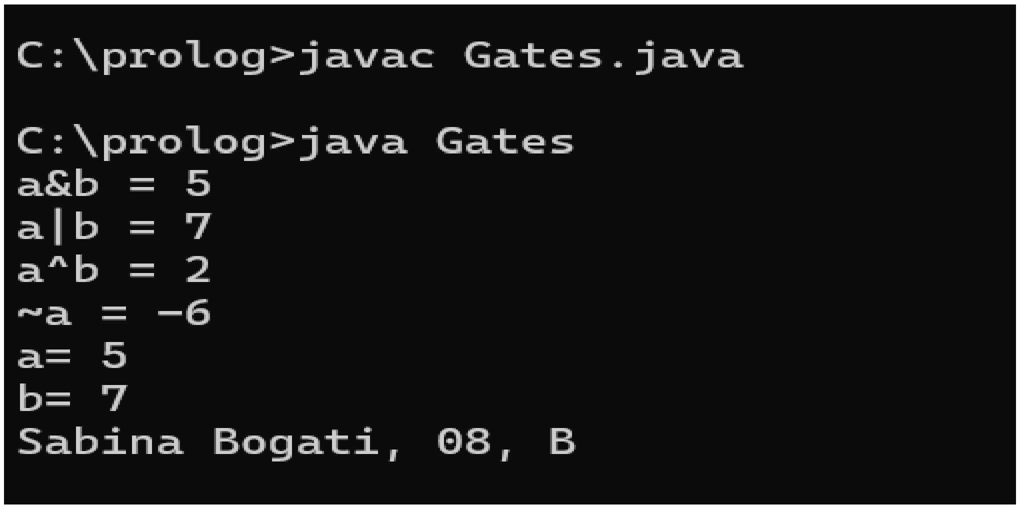
Objectives:

The objective of this lab aims at implementing AND, OR, NOT and XOR gate using java programming language.

Source code:

```
public class operators {
    public static void main(String[] args) {
        int a = 5;
        int b = 7;
        System.out.println("a&b = " + (a & b));
        System.out.println("a|b = " + (a | b));
        System.out.println("a^b = " + (a ^ b));
        System.out.println("~a = " + ~a);
        System.out.println("a= " + a);
        System.out.println("b= " + b);
        System.out.println("Sabina Bogati, 08, B");
    }
}
```

Output:



```
C:\prolog>javac Gates.java

C:\prolog>java Gates
a&b = 5
a|b = 7
a^b = 2
~a = -6
a= 5
b= 7
Sabina Bogati, 08, B
```