A
LAB REPORT OF
ARTIFICIAL INTELLIGENCE

Submitted By
Roshan Nepal
Roll No: 12093/20

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 BRIEF

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 logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations.

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:

```
language(java).
man(ram).
capital_of(Nepal,Kathmandu).
```

In this example, the first line states that java is a language, the second line states that ram is a man, and the third line states that Kathmandu is the capital of Nepal.

Variables in prolog:

In the 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 a uppercase letter, such as X or Y.

For example:

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

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

Query 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(france, X)?
```

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

History of prolog

Prolog, short for "Programming in Logic," is a high-level programming language primarily used for artificial intelligence and computational linguistics. It was conceived in the early 1970s by Alain Colmerauer, a French computer scientist, and his team at the University of Aix-Marseille. Prolog emerged from efforts to create a programming language that could efficiently handle symbolic reasoning and natural language processing tasks. The language was designed to leverage formal logic, specifically predicate logic, as its foundation, enabling it to represent and manipulate knowledge effectively.

The first implementation of Prolog was completed in 1972, with significant contributions from Robert Kowalski at the University of Edinburgh, who formalized the theoretical underpinnings of logic programming. Prolog introduced the concept of declarative programming, where programs are expressed in terms of facts and rules, and the system uses logical inference to derive conclusions. This unique paradigm made Prolog especially suitable for tasks like theorem proving, expert systems, and automated reasoning.

During the 1980s, Prolog gained prominence as a key language for artificial intelligence research and applications. It was notably adopted in Japan's Fifth Generation Computer Systems (FGCS) project, which aimed to develop advanced AI technologies. Although its popularity has declined compared to imperative languages, Prolog remains a vital tool in niche areas such as natural language processing, computational linguistics, and knowledge representation. Its contributions to the field of logic programming have left a lasting impact on the development of AI and computer science.

Use cases of prolog:

- **Artificial Intelligence (AI) and Expert Systems:** Prolog is widely used in AI for building expert systems, where it encodes domain-specific knowledge as facts and rules. It uses logical inference to deduce new information or solve problems. For instance, medical diagnosis systems can leverage Prolog to suggest potential diseases based on symptoms.

- **Natural Language Processing (NLP):** Prolog is well-suited for NLP tasks due to its ability to process symbolic and structured data. Applications include parsing natural language, creating grammar rules, building chatbots, and performing semantic analysis. Its declarative nature makes it ideal for defining linguistic rules and analyzing text.

- **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.

- **Database Querying and Data Analysis:** Prolog can work as a query language for databases, particularly those with complex relationships. It allows users to define queries in terms of logical rules and retrieve information based on conditions, akin to SQL but with more expressive power in some contexts.

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.

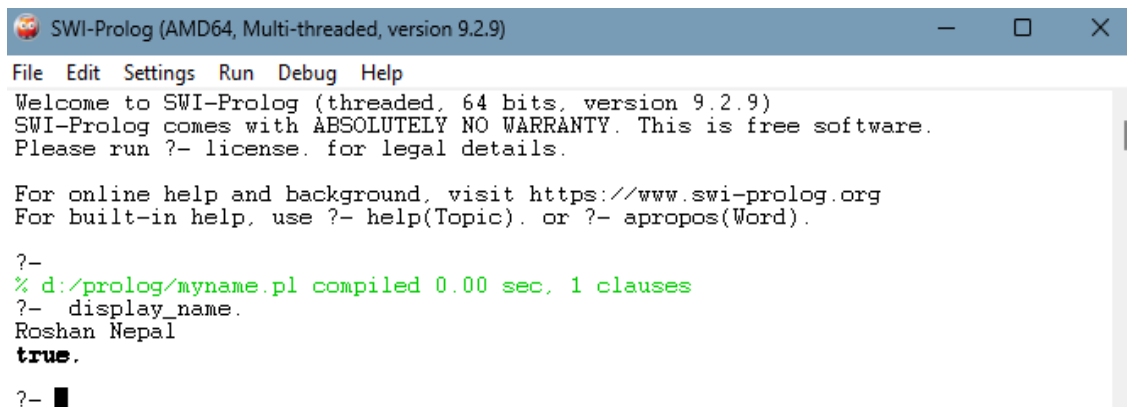# LAB II : WRITE A PROGRAM TO DISPLAY YOUR NAME IN PROLOG

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('Roshan Nepal').
```

Output

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
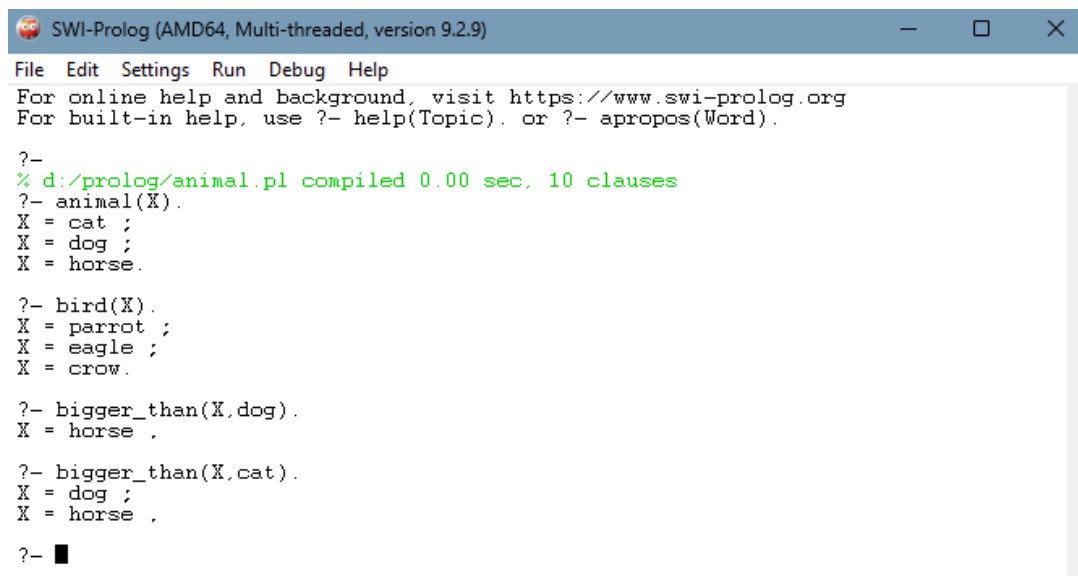
Objectives

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:

Observation:

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.

Conclusion:

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
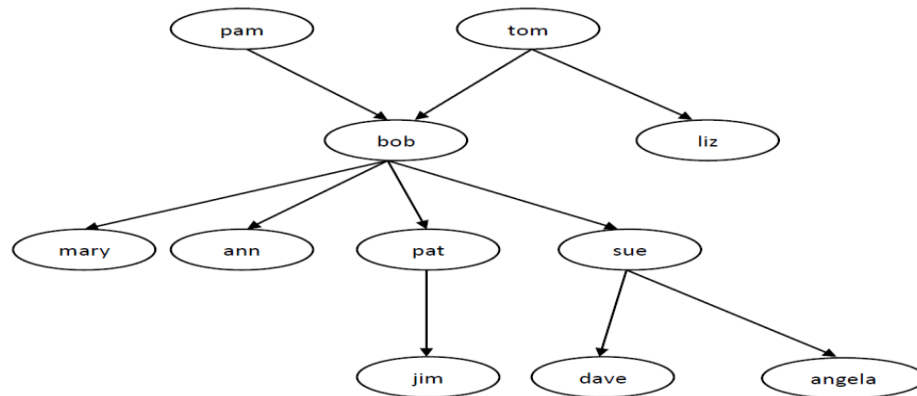
Objectives:

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?

```
% d:/prolog/family.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.
```

Observation:

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.

Conclusion:

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

Output:

i.   List all the male family members.

```
?-
% d:/prolog/family.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

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

 Enter a number: 4.

 Factorial of 4 is 24
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:

```
% d:/prolog/fibonacci.pl compiled 0.00 sec, 2 clauses
?-  fibonacciSequence.
Enter the position upto which you want to print Fibonacci Sequence : 5.

Fibonacci sequence upto  5 th term is :
0 1 1 2 3
 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

```
/-
% d:/prolog/monkey_banana_problem.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

```
% d:/prolog/hanoi_tower.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 the exam, they will be happy.

- Roshan studied for the exam.

- Sadip studied for the exam.

- Sameer studied for the exam.

- Sahil did not study for the exam.

- Kancha did not study for the exam.

Source code:

Output

i. Did Sameer pass?

```
% d:/prolog/exam.pl compiled 0.00 sec, 0 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.
```

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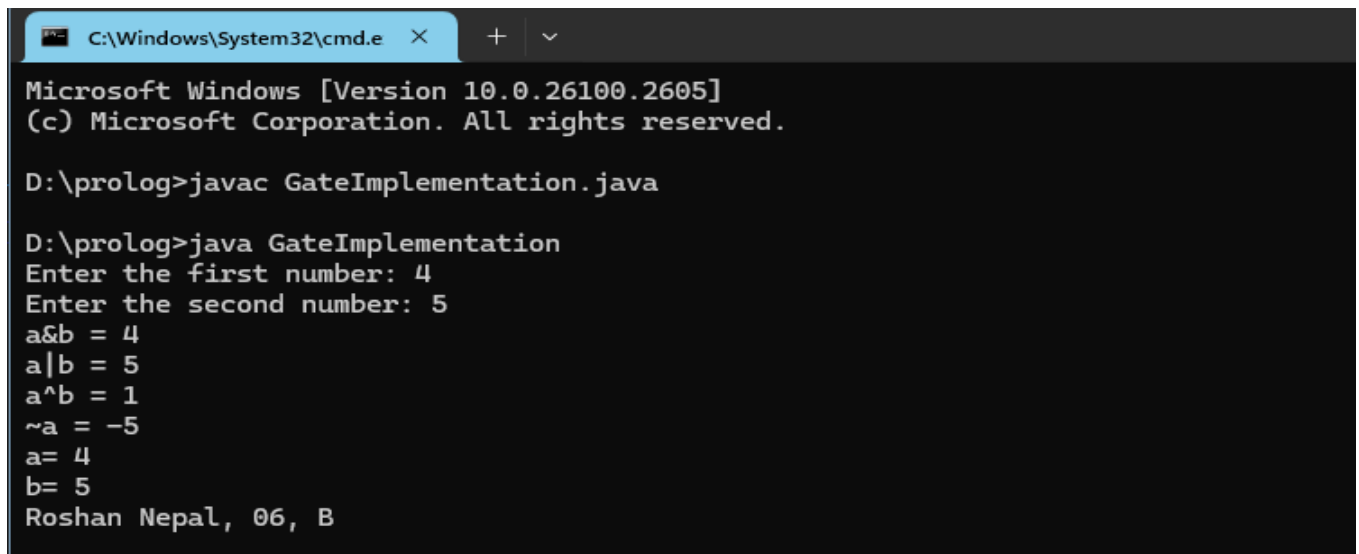
Objectives:

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

Source code:

```java
import java.util.Scanner;

public class GateImplementation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the first number: ");
        int a = sc.nextInt();
        System.out.print("Enter the second number: ");
        int b = sc.nextInt();
        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("Roshan Nepal, 06, B");

    }
}
```

Output

```
C:\Windows\System32\cmd.e   ×   +   ∨

Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

D:\prolog>javac GateImplementation.java

D:\prolog>java GateImplementation
Enter the first number: 4
Enter the second number: 5
a&b = 4
a|b = 5
a^b = 1
~a = -5
a= 4
b= 5
Roshan Nepal, 06, B
```