

---

## **EXPERIMENT NO.: 02**

**AIM:** Write a simple program using PROLOG as an AI Programming Language.

**OBJECTIVES:** Study of PROLOG

### **THEORY:**

PROLOG stands for Programming In Logic. It is language for symbolic, non-numeric computation. It is specially well suited for solving problems that involve objects and relation between objects. 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.

For example, it is an easy exercise in PROLOG to express spatial relationship between objects, such as the blue sphere is behind the green one. It is also easy to state a more general rule: if the object X is closer to the observer than object Y, and object Y is closer than Z, then X must be closer than Z. PROLOG can result about the spatial relationships and their consistency with respect to the general rule. Features like this makes the PROLOG a powerful language for Artificial Language AI, and non- numerical programming.

Prolog stands for programming in logic. In the logic programming paradigm, prolog language is most widely available. Prolog is a declarative language, which means that a program consists of data based on the facts and rules (Logical relationship) rather than computing how to find a solution. A logical relationship describes the relationships which hold for the given application.

To obtain the solution, the user asks a question rather than running a program. When a user asks a question, then to determine the answer, the run time system searches through the database of facts and rules.

Prolog features are 'Logical variable', which means that they behave like uniform data structure, a backtracking strategy to search for proofs, a pattern-matching facility, mathematical variable, and input and out are interchangeable.

To deduce the answer, there will be more than one way. In such case, the run time system will be asked to find another solution. To generate another solution, use the backtracking strategy. Prolog is a weakly typed language with static scope rules and dynamic type checking.

Prolog is a declarative language that means we can specify what problem we want to solve rather than how to solve it.

Prolog is used in some areas like database, natural language processing, artificial intelligence, but it is pretty useless in some areas like a numerical algorithm or instance graphics.

In artificial intelligence applications, prolog is used. The artificial intelligence applications can be automated reasoning systems, natural language interfaces, and expert systems. The expert system consists of an interface engine and a database of facts. The prolog's run time system provides the service of an interface engine.

A basic logic programming environment has no literal values. An identifier with upper case letters and other identifiers denote variables. Identifiers that start with lower-case letters denote data values. The basic Prolog elements are type less. The most implementations of prolog have been enhanced to include integer value, characters, and operations. The Mechanism of prolog describes the tuples and lists.

Functional programming language and prolog have some similarities like Hugs. A logic program is used to consist of relation definition. A functional programming language is used to consist of a sequence of function definitions. Both the logical programming and functional programming rely heavily on recursive definitions.

#### Syntax and Basic Fields:

In prolog, we declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (,). Their relation is expressed at the start and outside the parenthesis. Every fact/rule end with a dot (.). So, a typical prolog fact goes as follows:

Format: relation (entity1, entity2, .... k'th entity).

#### Example:

```
friends(raju, mahesh).  
singer(sonu).  
odd_number(5).
```

#### Explanation:

These facts can be interpreted as:  
raju and mahesh are friends.  
sonu is a singer.  
5 is an odd number.

Query Example:

A typical prolog query can be asked as:

Query 1: ?- singer(sonu).

Output: Yes.

Explanation: As our knowledge base contains the above fact, so output was 'Yes', otherwise it would have been 'No'.

Query 2: ?- odd\_number(7).

Output: No.

Explanation: As our knowledge base does not contain the above fact, so output was 'No'.

**PROGRAM:**

- Example 1:

friend(jin, james).

friend(jin, john).

likes(john, jin).

likes(james, john).

happy(X):-friend(X,Y),likes(Y,X).

- Example 2:

owns(jack , car(bmw)).

owns(john , car(chevy)).

owns(olivia , car(civic)).

owns(jane , car(chevy)).

sedan(car(bmw)).

sedan(car(civic)).

truck(car(chevy)).

- Example 3:

cat(fubby).

black\_spots(fubby).

dog(figaro).

white\_spots(figaro).

owns (mary, Pet):- cat(Pet), black\_spots(Pet).

loves (Who,What):- owns(Who ,What).

- Example 4:

woman(jia).

man(john).

healthy(john).

healthy(jia).

wealthy(john).

traveler(X) :- healthy(X), wealthy(X).

travel(X) :- traveler(X).

- Example 5:

/\* Facts \*/

male(jack).

male(oliver).

male(ali).

male(james).

male(simon).

male(harry).

female(helen).

female(sophie).

female(jess).

female(lily).

parent\_of(jack,jess).

parent\_of(jack,lily).

parent\_of(helen, jess).

parent\_of(helen, lily).

parent\_of(oliver,james).

parent\_of(sophie, james).

parent\_of(jess, simon).

parent\_of(ali, simon).

parent\_of(lily, harry).

parent\_of(james, harry).

/\* Rules \*/

father\_of(X,Y):- male(X), parent\_of(X,Y).

mother\_of(X,Y):- female(X), parent\_of(X,Y).

grandfather\_of(X,Y):- male(X), parent\_of(X,Z), parent\_of(Z,Y).

grandmother\_of(X,Y):- female(X), parent\_of(X,Z), parent\_of(Z,Y).

sister\_of(X,Y):- %(X,Y or Y,X)%female(X)

brother\_of(X,Y):- %(X,Y or Y,X)% male(X)

aunt\_of(X,Y):- female(X),

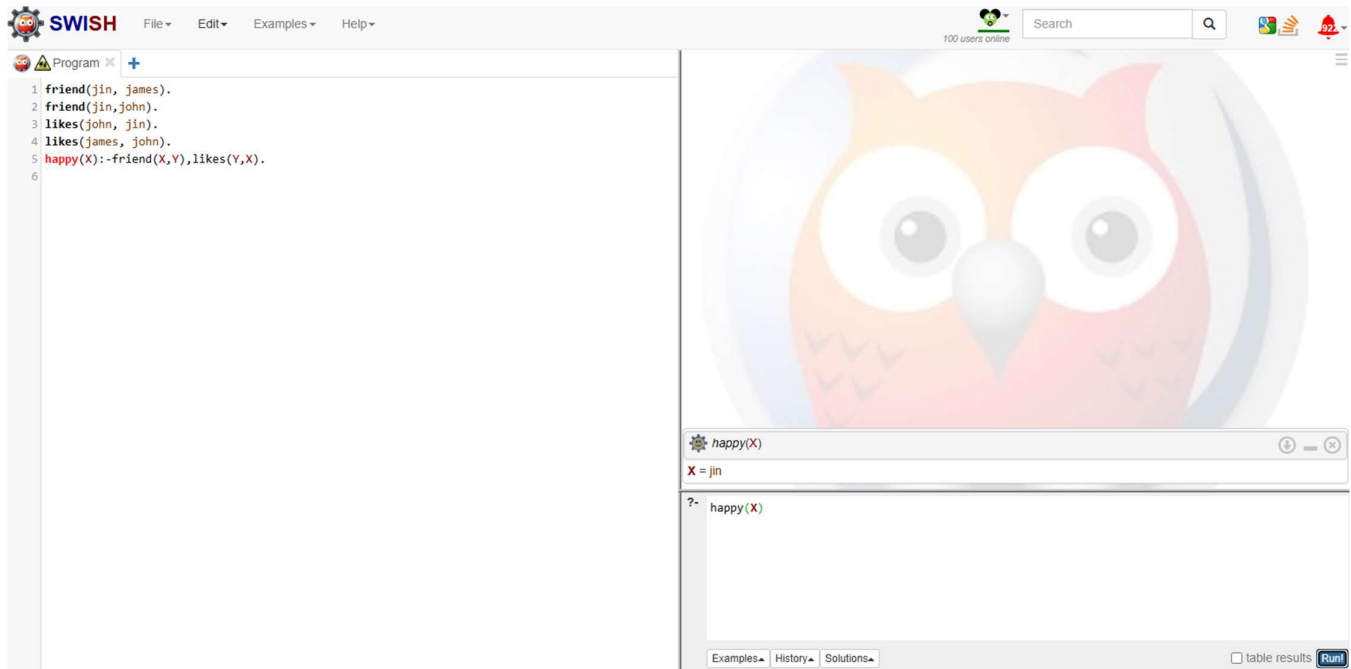
uncle\_of(X,Y):-parent\_of(Z,Y), brother\_of(Z,X).

ancestor\_of(X,Y):- parent\_of(X,Y).

ancestor\_of(X,Y):- parent\_of(X,Z), ancestor\_of(Z,Y).

## OUTPUT:

- Example 1:

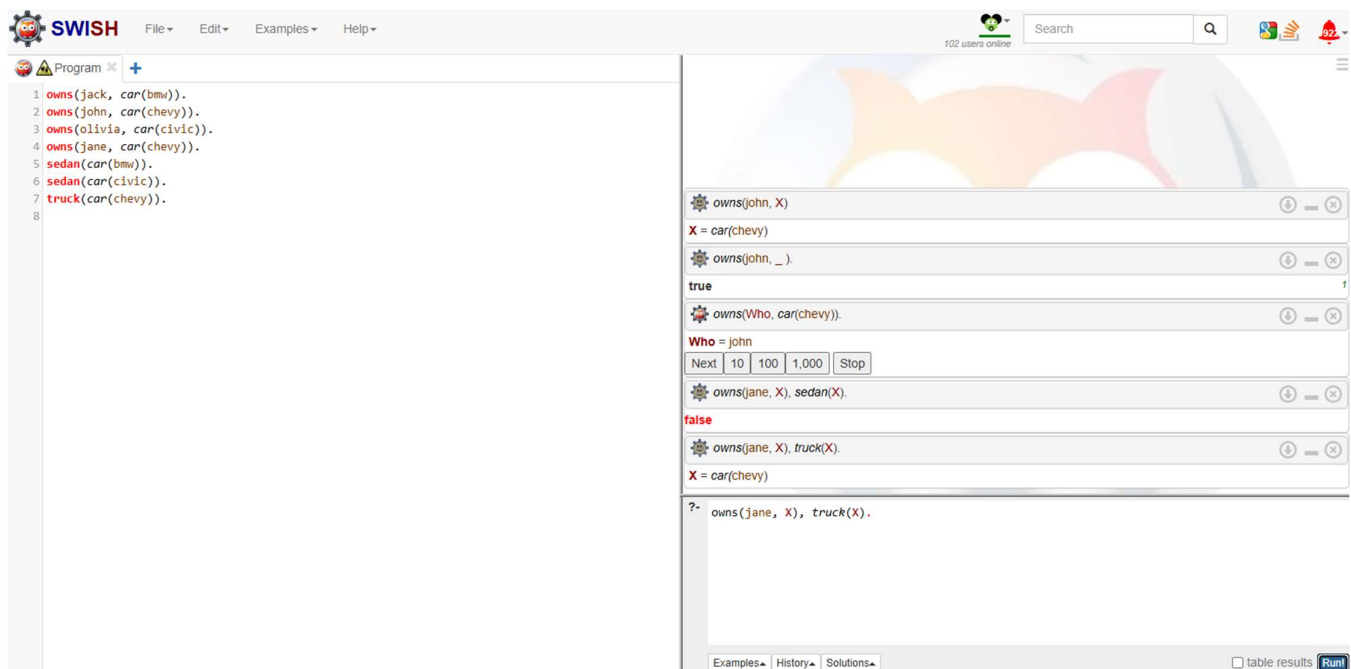


The screenshot shows the SWISH Prolog IDE interface. On the left, the 'Program' tab contains the following code:

```
1 friend(jin, james).  
2 friend(jin, john).  
3 likes(john, jin).  
4 likes(james, john).  
5 happy(X):-friend(X,Y),likes(Y,X).  
6
```

On the right, the execution results are displayed. The first query is `happy(X)`, which returns `X = jin`. Below this, the query `?- happy(X)` is shown, and the results are empty. The interface includes a search bar, a 'Run!' button, and a 'table results' checkbox.

- Example 2:



The screenshot shows the SWISH Prolog IDE interface. On the left, the 'Program' tab contains the following code:

```
1 owns(jack, car(bmw)).  
2 owns(john, car(chevy)).  
3 owns(olivia, car(civic)).  
4 owns(jane, car(chevy)).  
5 sedan(car(bmw)).  
6 sedan(car(civic)).  
7 truck(car(chevy)).  
8
```

On the right, the execution results are displayed. The first query is `owns(john, X)`, which returns `X = car(chevy)`. Below this, the query `owns(john, _)` returns `true`. The next query is `owns(Who, car(chevy))`, which returns `Who = john`. Below this, the query `owns(jane, X), sedan(X)` returns `false`. The final query is `owns(jane, X), truck(X)`, which returns `X = car(chevy)`. The interface includes a search bar, a 'Run!' button, and a 'table results' checkbox.

Class and Branch: TE AI-DS A  
Student Roll no.: A07  
Student Name: Niharika Seth

DOP: 09-08-2024

- Example 3:

The SWISH IDE interface displays a Prolog program in the left pane and its execution results in the right pane. The program defines facts about a cat named 'fubby' and a person named 'mary' who owns it. The results pane shows the execution of several queries, including 'listing(cat)', 'listing(owns)', 'loves(Who, What)', and 'owns(mary, \_)'. The background of the right pane features a large, stylized illustration of a cat's face.

```
1 cat(fubby).
2 black_spots(fubby).
3 dog(figaro).
4 white_spots(figaro).
5 owns(mary, Pet):- cat(Pet), black_spots(Pet).
6 loves(Who, What):-owns(Who, What).
```

listing(cat).  
cat(fubby).  
true  
listing(owns).  
owns(mary, Pet) :-  
cat(Pet),  
black\_spots(Pet).  
true  
loves(Who, What).  
What = fubby,  
Who = mary  
owns(mary, \_).  
true  
?- owns(mary, \_).

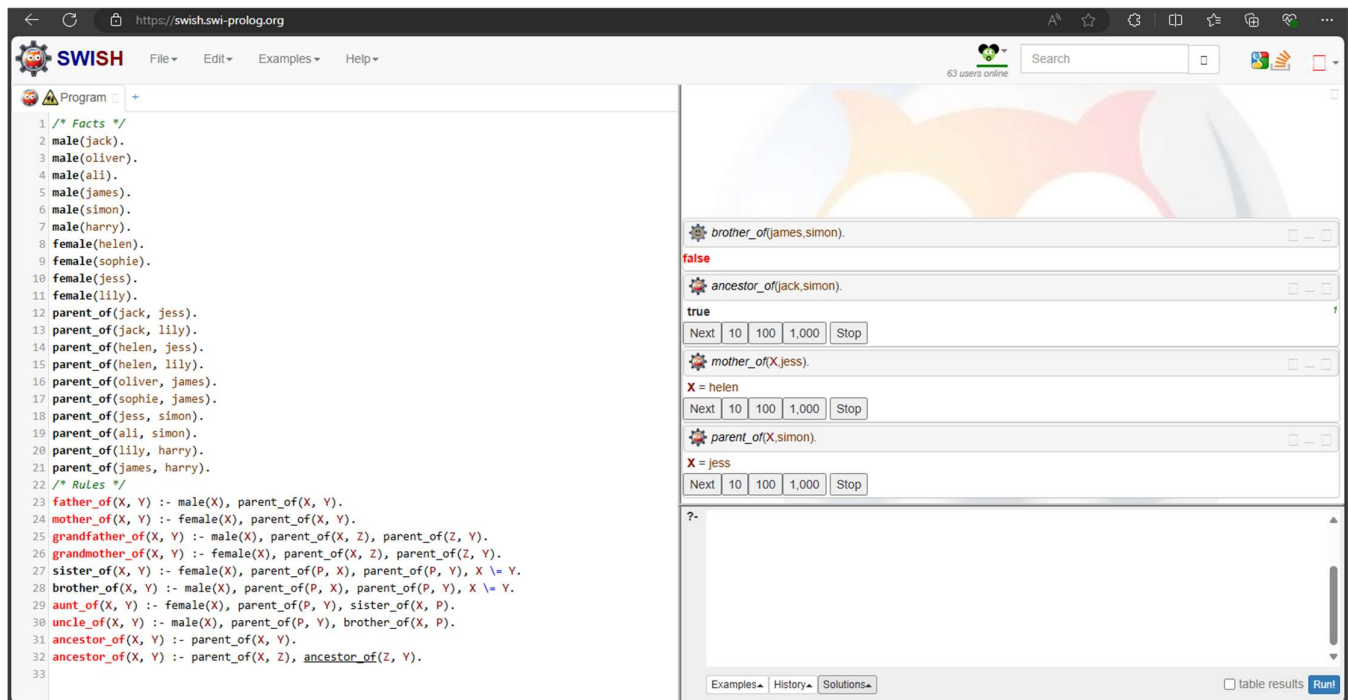
- Example 4:

The SWISH IDE interface displays a Prolog program in the left pane and its execution results in the right pane. The program defines facts about a woman named 'jia' and a man named 'john', and a rule for 'traveler(X)'. The results pane shows the execution of the query 'travel(X)', which returns 'X = john'. The background of the right pane features a large, stylized illustration of a cat's face.

```
1 woman(jia).
2 man(john).
3 healthy(john).
4 healthy(jia).
5 wealthy(john).
6 traveler(X) :- healthy(X), wealthy(X).
7 travel(X) :- traveler(X).
8
```

travel(X).  
X = john  
traveler(X).  
X = john  
?- traveler(X).

- Example 5:



The screenshot shows the SWISH Prolog environment. On the left, the Prolog program is defined with facts and rules. The facts define the gender and names of individuals: male(jack), male(oliver), male(al), male(james), male(simon), male(harry), female(helen), female(sophie), female(jess), and female(lily). The rules define family relationships: father\_of, mother\_of, grandfather\_of, grandmother\_of, sister\_of, brother\_of, aunt\_of, uncle\_of, and ancestor\_of. The right pane shows the execution of several queries. The query 'brother\_of(james,simon)' returns 'false'. The query 'ancestor\_of(jack,simon)' returns 'true'. The query 'mother\_of(X,jess)' returns 'X = helen'. The query 'parent\_of(X,simon)' returns 'X = jess'. The bottom of the right pane shows a search bar and a 'Run!' button.

```
1 /* Facts */
2 male(jack).
3 male(oliver).
4 male(al).
5 male(james).
6 male(simon).
7 male(harry).
8 female(helen).
9 female(sophie).
10 female(jess).
11 female(lily).
12 parent_of(jack, jess).
13 parent_of(jack, lily).
14 parent_of(helen, jess).
15 parent_of(helen, lily).
16 parent_of(oliver, james).
17 parent_of(sophie, james).
18 parent_of(jess, simon).
19 parent_of(al, simon).
20 parent_of(lily, harry).
21 parent_of(james, harry).
22 /* Rules */
23 father_of(X, Y) :- male(X), parent_of(X, Y).
24 mother_of(X, Y) :- female(X), parent_of(X, Y).
25 grandfather_of(X, Y) :- male(X), parent_of(X, Z), parent_of(Z, Y).
26 grandmother_of(X, Y) :- female(X), parent_of(X, Z), parent_of(Z, Y).
27 sister_of(X, Y) :- female(X), parent_of(P, X), parent_of(P, Y), X \= Y.
28 brother_of(X, Y) :- male(X), parent_of(P, X), parent_of(P, Y), X \= Y.
29 aunt_of(X, Y) :- female(X), parent_of(P, Y), sister_of(X, P).
30 uncle_of(X, Y) :- male(X), parent_of(P, Y), brother_of(X, P).
31 ancestor_of(X, Y) :- parent_of(X, Y).
32 ancestor_of(X, Y) :- parent_of(X, Z), ancestor_of(Z, Y).
33
```

brother\_of(james,simon).  
false

ancestor\_of(jack,simon).  
true  
Next 10 100 1,000 Stop

mother\_of(X,jess).  
X = helen  
Next 10 100 1,000 Stop

parent\_of(X,simon).  
X = jess  
Next 10 100 1,000 Stop

Examples History Solutions table results Run!

## OBSERVATION:

In this experiment, a simple PROLOG program was created to model a family tree using logical rules and facts to define relationships like parent, sibling and grandparent. The exercise highlighted PROLOG's strengths in logical reasoning and problem-solving, particularly through its querying and backtracking features. The experiment provided valuable insights into how PROLOG efficiently represents and infers complex relationships, showcasing its utility in AI programming.

## CONCLUSION:

In conclusion, PROLOG demonstrated the language's effectiveness in AI programming, particularly for tasks involving logical reasoning and knowledge representation. By defining relationships through facts and rules PROLOG efficiently handled complex queries and provided solutions through its powerful backtracking mechanism.