Introduction to logic programming (PROLOG)

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.

The first Prolog was 'Marseille Prolog', which is based on work by Colmerauer. The major example of fourth-generation programming language was prolog. It supports the declarative programming paradigm.

In 1981, a Japanese computer Project of 5th generation was announced. After that, it was adopted Prolog as a development language. In this tutorial, the program was written in the 'Standard' Edinburgh Prolog. Prologs of PrologII family are the other kind of prologs which are descendants of Marseille Prolog.

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 typeless. 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.

Applications of Prolog

The applications of prolog are as follows:

- Specification Language
- Robot Planning
- Natural language understanding
- Machine Learning
- Problem Solving
- Intelligent Database retrieval
- Expert System
- Automated Reasoning

Prolog system is straightforward. From one person to other person, the precise details of Prolog will vary. Prolog will produce a number of lines of headings in the starting, which is followed by a line. It contains just

?-

The above symbol shows the system prompt. The prompt is used to show that the Prolog system is ready to specify one or more goals of sequence to the user. Using a full stop, we can terminate the sequence of goals.

For example:

?- write('Welcome to Javatpoint'),nl,write('Example of Prolog'),nl.

nl indicates 'start a new line'. When we press 'return' key, the above line will show the effect like this:

Welcome to Javatpoint

Example of Prolog

yes

?- prompt shows the sequence of goal which is entered by the user. The user will not type the prompt. Prolog system will automatically generate this prompt. It means that it is ready to receive a sequence of goals.

The above example shows a sequence of goals entered by the user like this:

write('Welcome to Javatpoint'), write('Example of Prolog'), nl(twice).

Consider the following sequence of goals:

write('Welcome to Javatpoint'),nl,write('Example of Prolog'),nl.

The above sequence of goals has to succeed in order to be succeeded.

- write('Welcome to Javatpoint')

  On the screen of the user, Welcome to Javatpoint has to be displayed

- nl

  On the screen of the user, a new line has to be output

- write('Example of Prolog')

  On the screen of the user, Example of Prolog has to be displayed

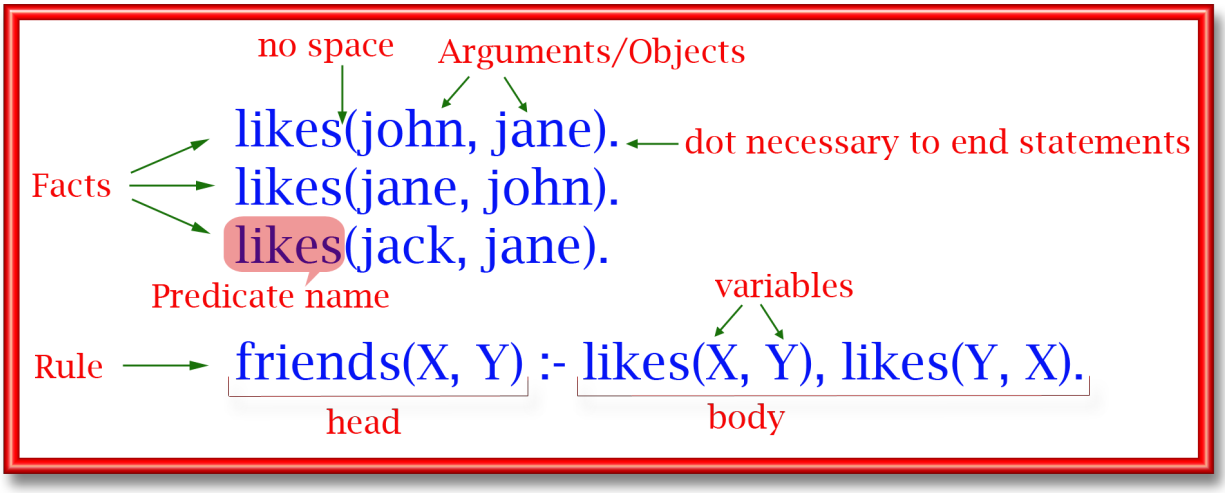
- nl

  On the screen of the user, a new line has to be output

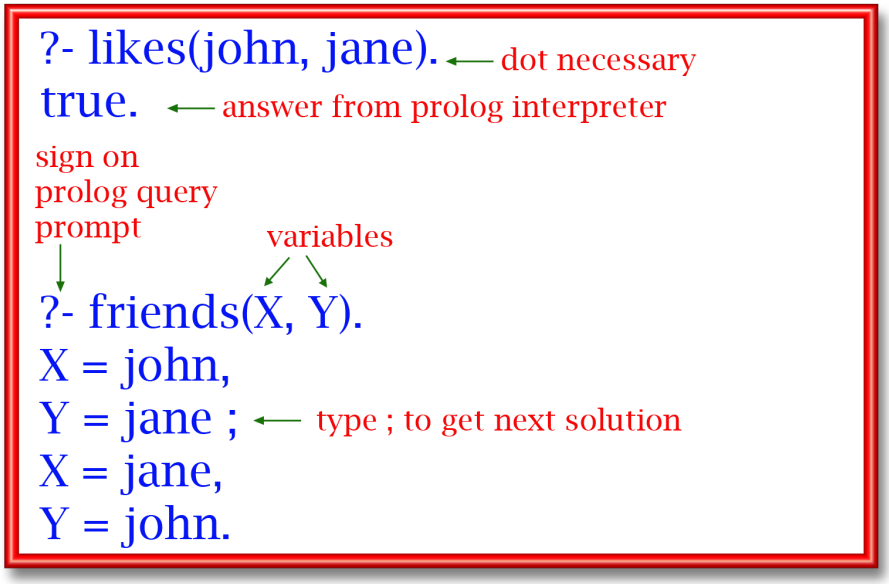
All these goals will simply be achieved by the Prolog system by outputting the line of text to the screen of the user. To show that the goals have succeeded, we will output yes.

The Prolog system predefined the meanings of nl and write. Write and nl are called as built-in predicates. 'Query' is a sequence of one or more goals. These goals are entered by the user at the prompt. In this tutorial, we are generally using the 'sequence of goals' term.

Program Window



Query Window



Example 1 : Below the food table shows the facts, rules, goals and their English meanings.

Facts

//English meanings

food(burger). // burger is a food

food(sandwich). // sandwich is a food

food(pizza). // pizza is a food

lunch(sandwich). // sandwich is a lunch

dinner(pizza). // pizza is a dinner

Rules

meal(X) :- food(X).

// Every food is a meal OR

Anything is a meal if it is a food

Queries / Goals

?- food(pizza).

// Is pizza a food?

?- meal(X), lunch(X).

// Which food is meal and lunch?

?- dinner(sandwich).

// Is sandwich a dinner?

Example 2 : Below student-professor relation table shows the facts, rules, goals and their english meanings.

Facts

//English meanings

studies(charlie, csc135).

// charlie studies csc135

studies(olivia, csc135).

// olivia studies csc135

studies(jack, csc131).

// jack studies csc131

studies(arthur, csc134).

// arthur studies csc134

teaches(kirke, csc135).

// kirke teaches csc135

teaches(collins, csc131).

// collins teaches csc131

teaches(collins, csc171).

// collins teaches csc171

teaches(juniper, csc134).

// juniper teaches csc134

Rules

professor(X, Y) :-

teaches(X, C), studies(Y, C).

// X is a professor of Y if X teaches C and Y studies C.

Queries / Goals

?- studies(charlie, What).

// charlie studies what? OR

What does charlie study?

?- professor(kirke, Students).

// Who are the students of professor kirke.

Example 3 – Arithmetic

Prolog arithmetic expression examples & exercise.

addition + multiplication * subtraction - division / power ^ mod

In prolog 'is' has a special functionality in evaluating arithmetic expressions. But with the condition that the expression should be on the right side of 'is' otherwise it will give an error. On Prolog Query Prompt :

?- X is 3+2. // expression on right side of 'is'

X = 5.

?- 3+2 is X. // expression on left side of 'is'

ERROR: is/2: Arguments are not sufficiently instantiated

?- X = 3+2. // just instantiate variable X to value 3+2

X = 3+2.

?- 3+2 = X.

X = 3+2.

?- X is +(3,2).

X = 5.

?- 5 is 3+2.

true.

?- 3+2 is 5.

false.

?- X is 3*2.

X = 6.

?- X is 3-2.

X = 1.

?- X is -(2,3).

X = -1.

?- X is 5-3-1.

X = 1.

?- X is -(5,3,1).

ERROR: is/2: Arithmetic: `(-)/3' is not a function

?- X is -(-(5,3),1).

X = 1.

?- X is 5-3-1.

X = 1

?- X is 3/5.

X = 0.6.

?- X is 3 mod 5.

X = 3.

?- X is 5 mod 3.

X = 2.

?- X is 5^3.

X = 125.

?- X is (5^3)^2.

X = 15625.

?- X = (5^3)^2.

X = (5^3)^2.

?- 25 is 5^2.

true.

?- Y is 3+2*4-1.

Y = 10.

?- Y is (3+2)*(4)-(1).

Y = 19.

?- Y is -(*(+(3,2),4),1).

Y = 19.

?- X is 3*2, Y is X*2.

X = 6,

Y = 12.