Date: 02/02/2024

Experiment - 1

Aim:

Study of PROLOG.

Theory:

Prolog is a logic programming language that is used to create artificial intelligence (AI). It is a free and open-source programming language that is available on many platforms.

Prolog is short for programming logic. It 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.

In Prolog, logic is expressed as relations (called as Facts and Rules). The programmer specifies a set of rules and facts about a problem domain, and then use those rules and facts to automatically infer solutions to problems.

- Facts − The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.
- Rules − Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as − grandfather(X, Y) :- father(X, Z), parent(Z, Y)

  This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

- Questions − And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

Prolog has many built-in features for AI programming, such as backtracking, data structures, and pattern matching.

Prolog was created around 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses.

Syntax:

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 ends with a dot (.).

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.

Key Features:

1. Unification: The basic idea is, can the given terms be made to represent the same structure.
2. Backtracking: When a task fails, prolog traces backwards and tries to satisfy previous task.
3. Recursion: Recursion is the basis for any search in program.

Some Applications of Prolog:

Prolog is used in various domains. It plays a vital role in automation system. Following are some other important fields where Prolog is used −

- Intelligent Database Retrieval
- Natural Language Understanding
- Specification Language
- Machine Learning
- Robot Planning
- Automation System

Date: 02/02/2024

- Problem Solving

PROLOG Data Types:

1. Char: Between a pair of single quotes, a character is encapsulated.
2. Integer: An integer between -32768 and 32767 that is a full number.
3. Real: A peculiar character that is either positive or negative, followed by numbers.
4. String: A collection of characters encased in a pair of double-quotes. Strings can have up to 255 characters in them.
5. Symbol: A combination of letters (A to Z or a to z), numerals (0 to 9) and the underscore(_) character.
6. Variables: A variable is a symbol that can have multiple values assigned to it at different stages of the program's execution.
7. Reserved terms: PROLOG features a few reserved words that should not be substituted for user-defined names.
8. Arithmetic Operators: The basic arithmetic operators in PROLOG are +, -, *, and /.
9. Relational Operators: PROLOG utilises the relational operators,=, =>, >=, >=. A relational operator in PROLOG can be either goal or subgoal. The relational operator (=) resembles an assignment operator in appearance.

Symbols:

Using the following truth-functional symbols, the Prolog expressions are comprised. These symbols have the same interpretation as in the predicate calculus.

English	**Predicate Calculus**	**Prolog**
If	-->	:-
Not	~	Not
Or	V	;
and	^	,

Date: 02/02/2024

Experiment - 2

Aim:

Write simple fact for the statements using PROLOG

a. Ram likes mango.
b. Seema is girl.
c. Bill likes Cindy.
d. Rose is red.
e. John owns gold.

Program:

likes(ram, mango).

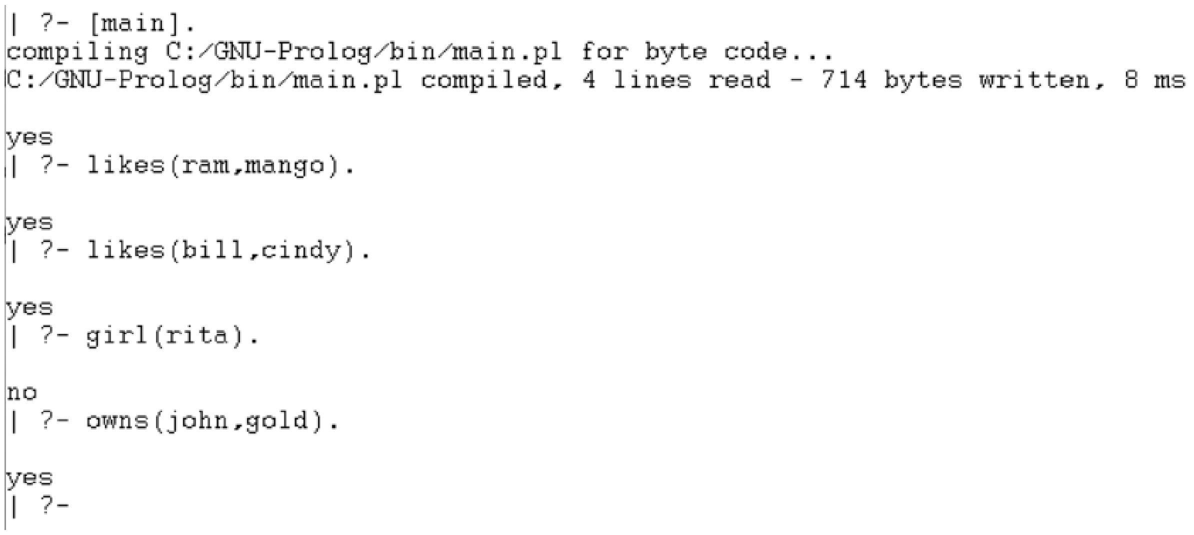
likes(bill, cindy).

girl(seema).

red(rose).

owns(john, gold).

Output:



```
| ?- [main].
compiling C:/GNU-Prolog/bin/main.pl for byte code...
C:/GNU-Prolog/bin/main.pl compiled, 4 lines read - 714 bytes written, 8 ms

yes
| ?- likes(ram,mango).

yes
| ?- likes(bill,cindy).

yes
| ?- girl(rita).

no
| ?- owns(john,gold).

yes
| ?-
```

Date: 09/02/2024

Experiment - 3

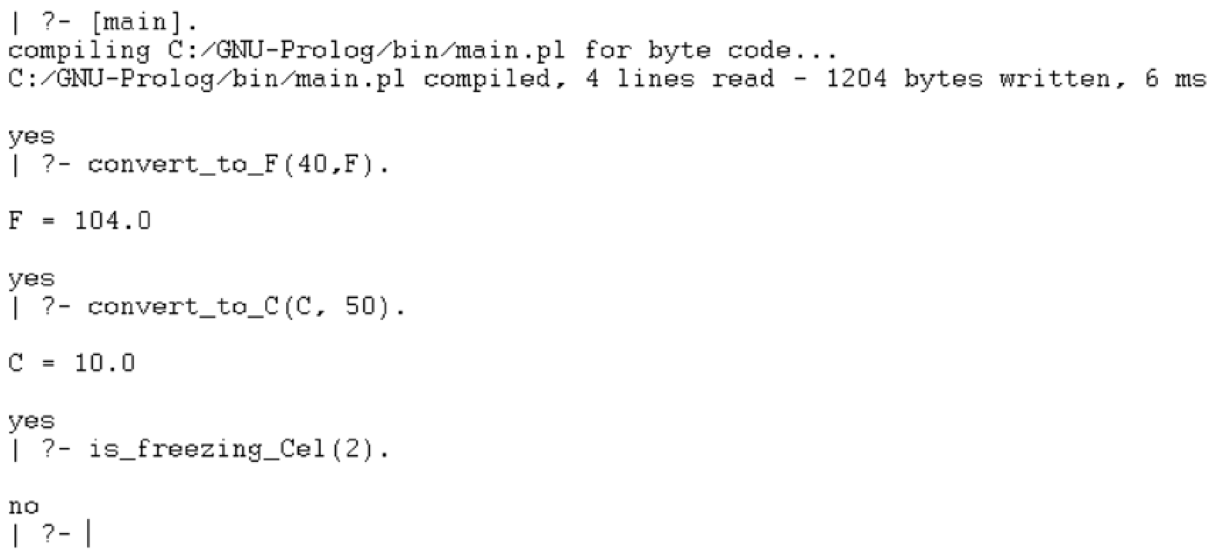
Aim:

Write predicates, one converts centigrade temperatures to Fahrenheit, the other checks if a temperature is below freezing point using PROLOG.

Program:

```
convert_to_F(C, F) :- F is (9*C)/5 + 32.

convert_to_C(C, F) :- C is (F-32)*5/9.

is_freezing_Cel(C) :- C<0.

is_freezing_Fah(F) :- F<32.
```

Output:



```
| ?- [main].
compiling C:/GNU-Prolog/bin/main.pl for byte code...
C:/GNU-Prolog/bin/main.pl compiled, 4 lines read - 1204 bytes written, 6 ms

yes
| ?- convert_to_F(40,F).

F = 104.0

yes
| ?- convert_to_C(C, 50).

C = 10.0

yes
| ?- is_freezing_Cel(2).

no
| ?- |
```

Date: 09/02/2024

Experiment - 4

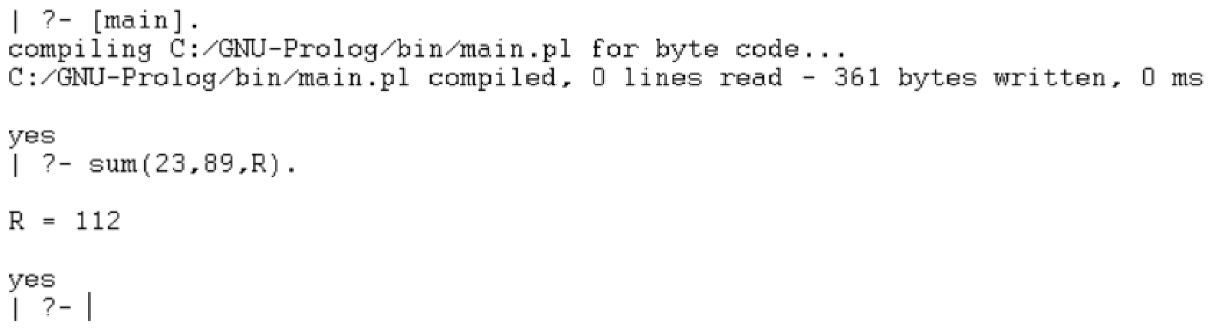
Aim:

Write program to calculate the sum of 2 numbers using variables.

Program:

```prolog
sum(X, Y, R) :- R is X+Y.
```

Output:



```
| ?- [main].
compiling C:/GNU-Prolog/bin/main.pl for byte code...
C:/GNU-Prolog/bin/main.pl compiled, 0 lines read - 361 bytes written, 0 ms

yes
| ?- sum(23,89,R).

R = 112

yes
| ?- |
```

Date: 16/02/2024

Experiment – 5

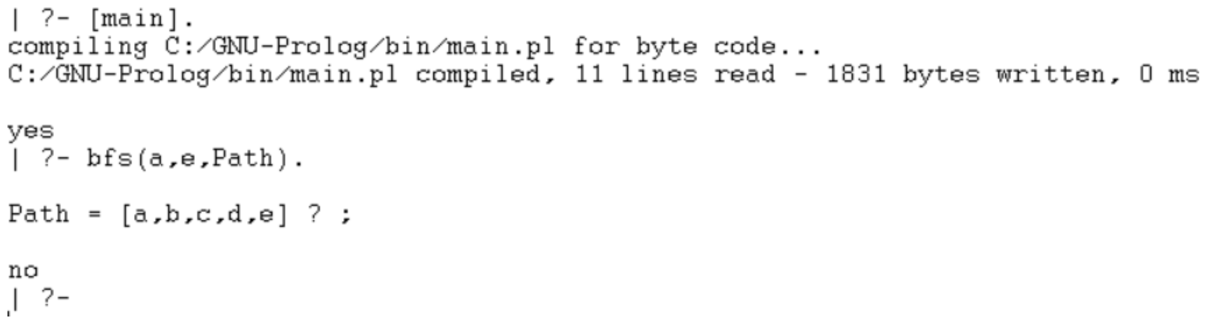
Aim:

Write a program to implement Breadth First Search Traversal.

Program:

```
edge(a, b).

edge(a, c).

edge(b, d).

edge(b, e).

edge(c, f).

edge(c, g).

goal(Node) :- Node = d.


bfs(Start, Goal, Path) :- bfs_queue([Start], Goal, Path).

bfs_queue([Node|_], Node, [Node]).

bfs_queue([Node|Rest], Goal, [Node|Path]) :- findall(Next, edge(Node, Next), Neighbours),
append(Rest, Neighbours, NewQueue), bfs_queue(NewQueue, Goal, Path).
```

Output:



```
| ?- [main].
compiling C:/GNU-Prolog/bin/main.pl for byte code...
C:/GNU-Prolog/bin/main.pl compiled, 11 lines read - 1831 bytes written, 0 ms

yes
| ?- bfs(a,e,Path).

Path = [a,b,c,d,e] ? ;

no
| ?-
```

Date: 16/02/2024

Experiment - 6

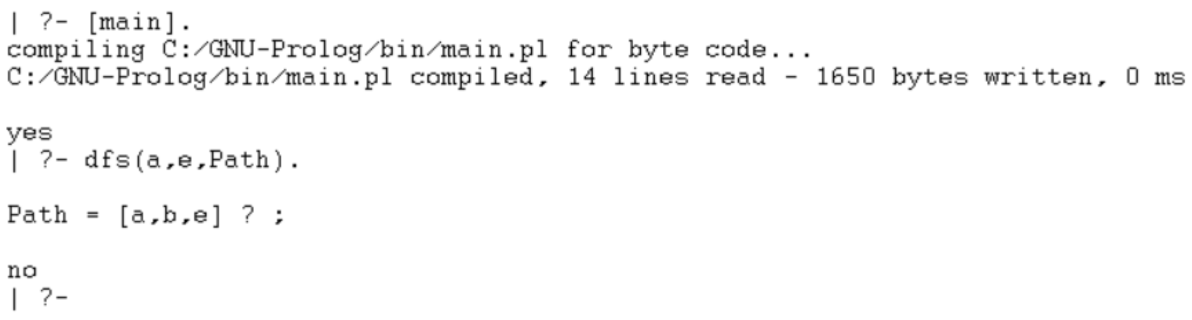
Aim:

Write a program to implement Depth First Search Traversal.

Program:

```
edge(a, b).
edge(a, c).
edge(b, d).
edge(b, e).
edge(c, f).
edge(c, g).

dfs(Node, Goal, Path) :-
   dfs_stack(Node, Goal, [Node], Path).
dfs_stack(Node, Node, _, [Node]) :- !.
dfs_stack(Node, Goal, Visited, [Node | Path]) :-
   edge(Node, Next),
   \+ member(Next, Visited),
   dfs_stack(Next, Goal, [Next | Visited], Path).
```

Output:



```
| ?- [main].
compiling C:/GNU-Prolog/bin/main.pl for byte code...
C:/GNU-Prolog/bin/main.pl compiled, 14 lines read - 1650 bytes written, 0 ms

yes
| ?- dfs(a,e,Path).

Path = [a,b,e] ? ;

no
| ?-
```