AI LAB-1

Objective: Study of Facts, Objects, Predicates and Variables.

Description:

The name Prolog was taken from the phrase "*programming in logic*." Prolog is unique in its ability to infer facts and conclusions from other facts. Mainly Prolog is used to build expert systems. Knowledge engineers design and build the system that use Prolog. The knowledge engineer listens to experts in a particular domain, abstracting the subjective methods used by human mind and defining an objective system of formal reasoning that can be supported by a Prolog structure.
Prolog program consists of a collection of knowledge about specific subject. This collection is called a database and database is expressed in facts and rules.

Facts:
A **fact** is just what it appears to be --- a fact. A fact in everyday language is often a proposition like ``It is sunny." or ``It is summer." In Prolog such facts could be represented as follows:

```
'It is sunny'.
'It is summer'.
```

Prolog describes facts as symbolic relationships. for example, if the right speaker in your stereo system is not omitting sound ,you can express this in English as

The right speaker is dead.

This fact can be expressed in Prolog as

```
is(right_speaker,dead).
```

This factual expression is also called ***clause.***

Objects:
An object is the name of an element of certain type. It represents the entity or a property of an entity in the real world.
In the above example right_speaker and dead both are objects.

- **Types of Objects**:
  There are six types of objects.

Char	Single char (enclosed between single quotation marks)
Integer	Integer from -32,768 to32,767
Real	Floating-point number(1e^{-307} to 1e^{308})
String	Character sequence(enclosed between double quotation marks)

• Symbol	Char sequence of letters, numbers and underscores with the first Character a lowercase letter.
• File	Symbolic file name

Predicates:-

A Predicate is a function with a value of true and false. Predicate express a property or relationship.

e.g.

is (right_speaker,dead)

The word before parentheses is the name of the ***relation***. The elements within the parentheses are the **arguments of the predicate**, which **may be objects or variables**.

❖ **Sample Program:**

```
domains
        disease, indication=symbol
predicates
        symptom (disease, indication)
clauses
        symptom(chicken_pox,high_fever).
        symptom(chicken_pox,chills).
        symptom(flu,chills).
        symptom(cold,runny_nose).
        symptom(flu,runny_nose).
```

I/P:

```
        Goal: symptom(cold,runny_nose) ↵
```

O/P: Turbo Prolog respond with True and prompt for another goal:

```
        True
        Goal:
```

Code-1

Variable:

A variable must begin with capital letter and may be from 1 to 250 characters long. Except from first character in the name, you may use uppercase or lowercase letters, digits or the underline character.

e.g.

Change the Goal for code1

I/P:

```
        Goal: symptom(Disease,runny_nose)
```

O/P: Turbo Prolog will respond

```
        Disease=cold
```

Disease=flu
2 Solutions
Goal:

- **Types of variables:**
  Bound variable: If the variable has a value at a particular time, it is said to be bound.
  Free variable: If the variable does not have a value at a particular time, it is said to be free. In above example program start with a free variable, Disease.

Anonymous variables:
Sometimes you may wish Prolog to ignore value of one or two arguments when determining a goal's failure or success. To accomplish this express the argument as an underline.
Change the Goal for code 1.
I/P:

Goal:symptom(_,chills)
O/P:
True
Goal:

Because there is no variable, no binding;
if Prolog can match the relation name and the last argument, the goal succeeds.

Compound Goals:
In Compound goal all the specified condition must succeed for the goal to succeed. Suppose the patient has a runny nose & mild body ache. This could be expressed as this Prolog goal.

Change the goal for code 1.
I/P: Goal:symptom(Disease.mild_body_ache),symptom(Disease,runny_nose)
O/P: Disease = cold
1 solution
Goal:

Backtracking:
Backtracking is a process. When a subgoal fails, the Prolog system traces its steps backwards to the previous goal and tries to *resatisfy* it.
This means that all variables that were bound to a value when that goal was satisfied are now made free again. Then the Prolog system tries to resatisfy that goal by matching with the *next clause* starting at the clause just after the one that matched the subgoal.
This continues until either the subgoal is satisfied, or until the program database has been exhausted, at which point Prolog backtracks yet again to the subgoal that was before the current subgoal.