BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Rishabh Santosh Shenoy
UID no.	2023300222
Experiment No.	2

AIM:	Prolog Program using knowledge base and inference rules
Program 1	
PROBLEM STATEMENT :	Prolog Program using knowledge base and inference rules for Pharmacy management system - implementing Expert system , propositional logic , predicate logic , Forward chaining , Backward chaining and Resolution.
THEORY:	Prolog (Programming in Logic) is a declarative language widely used in Artificial Intelligence and Soft Computing (AISC) for knowledge representation and reasoning. Unlike procedural languages, Prolog focuses on defining relationships (facts) and inference rules, while the interpreter automatically applies logical reasoning to answer queries. It is based on predicate logic and uses unification and backtracking to derive solutions. Prolog supports expert systems, natural language processing, and automated decision-making. Its strength lies in forward chaining, backward chaining, and resolution techniques, enabling efficient problem-solving where reasoning and inference are essential.
EXPERT SYSTEM:	An Expert System is a computer program that uses knowledge and inference rules to solve complex problems in a specific domain. It mimics human decision-making by applying reasoning over a knowledge base. In this experiment, the expert system is used for Pharmacy Management, where the system can store medicines, check stock, restock items, and sell medicines.
PROPOSITIONAL LOGIC:	Propositional Logic deals with statements that can be true or false. In the Pharmacy Management System, propositional logic can represent facts such as: 1)"Medicine Paracetamol exists." 2)"Stock of Ibuprofen is greater than 0." For example, a proposition P: medicine(paracetamol) is true if paracetamol exists in the knowledge base, otherwise false.

PREDICATE LOGIC:	Predicate Logic extends propositional logic by using quantifiers and relations between objects. In Prolog, it is represented using facts and rules. Example: medicine(paracetamol, 59, 100). This means "Paracetamol is a medicine with price Rs.59 and quantity 100." Predicates allow us to define rules such as selling, restocking, and deleting medicines.
FORWARD CHAINING:	Forward chaining is a reasoning technique that starts from known facts in the knowledge base and applies inference rules to derive new facts until a goal is reached. It is data-driven because it begins with available information and moves forward. In our program: Facts like medicine(paracetamol, 59, 100) already exist. When we execute a rule like add_medicine/3, a new fact is asserted into the knowledge base (assertz(medicine(Name, Price, Quantity))). These new facts can trigger other rules. For example: Adding a medicine automatically allows it to be sold or restocked later. After selling (sell_medicine/2), the stock quantity fact is updated in the knowledge base.
BACKWARD CHAINING:	Backward chaining starts with a goal (query) and works backward to see if the facts and rules in the knowledge base can satisfy that goal. It is goal-driven because the reasoning begins with what we want to prove. In our program: When a user queries check_stock(paracetamol), the system tries to satisfy the goal by checking if medicine(paracetamol, Price, Quantity) exists. If found, the query succeeds and outputs the stock. If not found, it searches alternative clauses until it either succeeds or fails.
RESOLUTION	Resolution is a rule of inference used in logic systems to derive conclusions by eliminating contradictions. In Prolog, resolution works through unification + backtracking. Unification: Matching a goal with a fact/rule by substituting variables. Backtracking: If one rule fails, Prolog goes back and tries another rule until success or all possibilities are exhausted.

In our program:
Suppose we query:
sell_medicine(amoxicillin, 40).

Step 1: Prolog unifies with sell_medicine(Name, SellQty) where Name = amoxicillin, SellQty = 40.
Step 2: It tries rule 1:

```
sell_medicine(Name, SellQty) :-
   medicine(Name, Price, CurrentQty),
   CurrentQty >= SellQty, ...
```

But CurrentQty = 30 (from fact) is not ≥ 40, so this rule fails.

Step 3: Prolog backtracks to next clause:

```
sell_medicine(Name, SellQty) :-
   medicine(Name, _, CurrentQty),
   CurrentQty < SellQty, ...
```

This succeeds because 30 < 40.

Step 4: Output → "Not enough stock."

PROGRAM:

```
:- dynamic medicine/3.

medicine(paracetamol, 59, 100).
medicine(ibuprofen, 75, 50).
medicine(amoxicillin, 139, 30).

add_medicine(Name, Price, Quantity) :-
   \+ medicine(Name, _, _),
   assertz(medicine(Name, Price, Quantity)),
   write('Medicine added successfully.'), nl.

add_medicine(Name, _, _) :-
   medicine(Name, _, _),
   write('Medicine already exists.'), nl.

view_all_medicines :-
   write('--- Medicine List ---'), nl,
   (   medicine(Name, Price, Quantity),
```

```
        format('~w - Rs.~w - Qty: ~w~n', [Name, Price, Quantity]),
        fail
    ;   true
    ).

check_stock(Name) :-
    write('Check stock pantry'),nl.
    medicine(Name, Price, Quantity),
    format('~w: Rs.~w, Stock: ~w~n', [Name, Price, Quantity]), !.

check_stock(Name) :-
    \+ medicine(Name, _, _),
    format('Medicine "~w" not found.~n', [Name]).

sell_medicine(Name, SellQty) :-
    medicine(Name, Price, CurrentQty),
    CurrentQty >= SellQty,
    NewQty is CurrentQty - SellQty,
    retract(medicine(Name, Price, CurrentQty)),
    assertz(medicine(Name, Price, NewQty)),
    Total is Price * SellQty,
    format('Sold ~w units of ~w for Rs.~2f.~n', [SellQty, Name, Total]), !.

sell_medicine(Name, SellQty) :-
    medicine(Name, _, CurrentQty),
    CurrentQty < SellQty,
    write('Not enough stock.'), nl, !.

sell_medicine(Name, _) :-
    \+ medicine(Name, _, _),
    format('Medicine "~w" not found.~n', [Name]).

restock_medicine(Name, AddQty) :-
    medicine(Name, Price, CurrentQty),
    NewQty is CurrentQty + AddQty,
    retract(medicine(Name, Price, CurrentQty)),
    assertz(medicine(Name, Price, NewQty)),
    write('Medicine restocked.'), nl, !.
```

```
restock_medicine(Name, _) :-
    \+ medicine(Name, _, _),
    format('Medicine "~w" not found.~n', [Name]).

delete_medicine(Name) :-
    retractall(medicine(Name, _, _)),
    format('Medicine "~w" deleted (if it existed).~n', [Name]).
```

INPUT:

```
add_medicine(protein, 300, 10),
view_all_medicines,
check_stock(paracetamol),
sell_medicine(protein, 5),
view_all_medicines,
restock_medicine(ibuprofen,30) ,
view_all_medicines
```

Examples History Solutions table results Run!

OUTPUT:

```
Medicine added successfully.
--- Medicine List ---
paracetamol - Rs.59 - Qty: 100
ibuprofen - Rs.75 - Qty: 50
amoxicillin - Rs.139 - Qty: 30
protein - Rs.300 - Qty: 10
Check stock pantry
Sold 5 units of protein for Rs.1500.00.
--- Medicine List ---
paracetamol - Rs.59 - Qty: 100
ibuprofen - Rs.75 - Qty: 50
amoxicillin - Rs.139 - Qty: 30
protein - Rs.300 - Qty: 5
Medicine restocked.
--- Medicine List ---
paracetamol - Rs.59 - Qty: 100
amoxicillin - Rs.139 - Qty: 30
protein - Rs.300 - Qty: 5
ibuprofen - Rs.75 - Qty: 80
true
```

1

Next 10 100 1,000 Stop

CONCLUSION:	I have understood that this experiment is very important in the context of Artificial Intelligence and Soft Computing (AISC) because it demonstrates how logical reasoning and knowledge representation can be applied to real-world domains. Prolog, being a declarative language, is especially useful for implementing expert systems where decisions are taken based on predefined facts and inference rules rather than procedural steps. The concepts of propositional logic, predicate logic, forward chaining, backward chaining, and resolution are fundamental building blocks of AI reasoning systems, and this experiment shows their application in a practical and structured way. I have also understood through the Pharmacy Management case study how Prolog can be used to store, retrieve, and update knowledge dynamically. I learned how facts such as medicine availability are represented in a knowledge base and how queries are resolved through logical inference. This helped me gain clarity on expert system design and its usefulness in automating decision-making processes.