UNIT - I

What are building blocks of prolog?

In Prolog, the building blocks of the language include the following fundamental components:

1. Facts

Facts represent statements about the world that are unconditionally true.
Syntax: predicate(argument1, argument2, ...).
Example:

```
parent(john, mary).
```

(This means John is a parent of Mary.)

2. Rules

Rules define relationships based on conditions.
A rule consists of a head (conclusion) and a body (conditions).
Syntax:

```
head :- body.
```

Example:

```
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

(X is a grandparent of Y if X is a parent of Z and Z is a parent of Y.)

3. Queries (Goals)

Queries are used to ask questions to the Prolog system.
Syntax:

```
?- predicate(argument1, argument2, ...).
```

Example:

```
?- parent(john, mary).
```

(Asks whether John is a parent of Mary.)

4. Atoms

Atoms are constant symbolic values (like names).
They start with a lowercase letter or are enclosed in single quotes .
Examples: 'john', 'apple', 'New York'.

5. Variables

Variables start with an uppercase letter or an underscore.
They act as placeholders for values.
Example:

```
parent(john, X).
```

(Finds all children of John.)

6. Structures (Compound Terms)

Structures represent complex objects with multiple components.

Example:
date(2024, march, 23).
(Represents a date with year, month, and day.)

7. Lists
Lists are used to store collections of elements.
Syntax: [Element1, Element2, ...]
Example:
[apple, banana, cherry].
The head and tail notation is:
[H | T].
(H is the first element, T is the rest of the list.)

8. Arithmetic and Comparison Operators
Prolog supports arithmetic operations like +,-, *, /, mod.
Comparisons: =, \= (\ is NOT), >, <, >=, =<.
Example:
X is 5 + 3.

9. Control Structures
Prolog provides control structures like:
, (AND operator)
; (OR operator)
not or \ (Negation)
Example:
happy(X) :- rich(X), healthy(X).

These are the core building blocks of Prolog, enabling it to function as a logic programming language.

Write an example program defining family relationships?

We can define the facts with a predicate parent that defines parent-child relationships

parent(john, mary).
parent(john, paul).
parent(susan, mary).
parent(susan, paul).
parent(mary, lisa).
parent(mary, james).
parent(paul, emma).
parent(paul, mark).

The program can define some more facts that are used to define rules.:-
Facts: Gender information

```
male(john).
male(paul).
male(james).
male(mark).
female(susan).
female(mary).
female(lisa).
female(emma).
```

The program can be further extended by rules :-

Rule: Defining the mother relationship

```
mother(X, Y) :-
    parent(X, Y),
    female(X).
```

Rule: Defining the father relationship

```
father(X, Y) :-
    parent(X, Y),
    male(X).
```

Rule: Defining the sibling relationship

```
sibling(X, Y) :-
    parent(Z, X),
    parent(Z, Y),
    X \= Y.
```

Rule: Defining the grandparent relationship

```
grandparent(X, Y) :-
    parent(X, Z),
    parent(Z, Y).
```

Rule: Defining the ancestor relationship

```
ancestor(X, Y) :-
    parent(X, Y).
ancestor(X, Y) :-
    parent(X, Z),
    ancestor(Z, Y).
```

What is recursive rule definition?

Recursive Rule Definition in Prolog
A recursive rule definition in Prolog is a rule that calls itself either directly or indirectly to solve a problem. Recursion is a key feature of Prolog and is commonly used for navigating hierarchical structures (like family trees).

Example : Recursive Ancestor Relationship
Here's a recursive definition of an ancestor in a family tree:

```
% Facts: Parent-child relationships
parent(john, mary).
parent(mary, lisa).
parent(lisa, emma).

Base case: A direct parent is an ancestor
ancestor(X, Y) :- parent(X, Y).

Recursive case: If X is a parent of Z and Z is an ancestor of Y, then X is an ancestor of Y
ancestor(X, Y) :-
   parent(X, Z),
   ancestor(Z, Y).
```

The base case states that if 'X' is a direct parent of 'Y', then 'X' is an ancestor of 'Y'.
The recursive case states that if 'X' is a parent of 'Z', and 'Z' is an ancestor of 'Y', then 'X' is also an ancestor of 'Y'.

to find all ancestors of emma, we can pass a query

```
?- ancestor(X, emma).
Output:
X = lisa ;
X = mary ;
X = john.
```

Key Points About Recursion in Prolog
1. Always define a base case to prevent infinite loops.
2. Recursive cases should reduce the problem size in each step.
3. Recursion is useful for hierarchical relationships (e.g., trees) and iterating over lists.

How Prolog answers questions?

Prolog answers questions using a process called goal resolution, based on backtracking and unification.

1. Unification (Matching)

- Prolog checks whether a query matches a fact or the head of a rule.
- If a match is found, Prolog binds variables to values.

Example:

if a fact declared as : parent(john, mary).

and if a query passed as ;

?- parent(john, X). % Prolog searches for X

X = mary.

That is Prolog unifies 'X' with 'mary'.

2. Rule Resolution (Using Rules to Derive Answers)

If a direct fact doesn't match, Prolog looks for a rule that can help answer the question.

Example:

if the facts declare are :

```
parent(john, mary).
parent(mary, lisa).
```

rule declared is :

```
grandparent(X, Y) :-
    parent(X, Z),
    parent(Z, Y).
```

?- grandparent(john, X).

Prolog's Steps:

a. Breaks down the rule:
   grandparent(john, X) :- parent(john, Z), parent(Z, X).
b. Finds a match:
   parent(john, mary) => So, Z = mary.
c. Continues with the second goal:
   parent(mary, X) => Finds X = lisa.

so it gives Output:

X = lisa.

3. Backtracking (Finding Alternative Answers)

If Prolog finds one answer, it backtracks to find more.

If a goal fails, it undoes previous variable bindings and tries the next possible solution.

Example:
if the facts declared are :
parent(john, mary).
parent(john, paul).

if a query passed :
?- parent(john, X).
Prolog's Steps:
1. First answer: `X = mary.`
2. Backtracks for more: `X = paul.`

Output:
X = mary ; (we put a ; to get further more answers.
X = paul.

4. Recursion in Answers
- When a query involves recursive rules , Prolog keeps searching until the base case is met.

Example:
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

?- ancestor(john, lisa).
Prolog's Steps:
a.1. ancestor(john, lisa) => parent(john, Z), ancestor(Z, lisa).
b. Z = mary (from parent(john, mary))
c. ancestor(mary, lisa) => parent(mary, lisa).
d. Base case met! Answer: X = john.

Output:
X = john.

5. Handling Yes/No Queries
If a query has a definite answer , Prolog responds with:
- 'true.' (if it can prove the statement)
- 'false.' (if it fails)

Example:
```prolog
?- parent(john, mary).
Output: true.
```

?- parent(paul, mary).
Output: false.

overall the prolog systems answers the queries in following ways :

a) Unification :- Matches facts and rules.
b) Rule Resolution :- Uses rules to derive answers.
c) Backtracking :- Finds all possible answers.
d) Recursion :- Solves hierarchical problems like family trees.
e) Handling true/false

What is declarative and procedural meaning of programs?

Prolog is a declarative programming language, meaning that a program describes what is true rather than specifying how to compute it. The declarative meaning of a Prolog program is based on logic and states relationships between objects using facts and rules. For example, in a family tree, a rule like `ancestor(X, Y) :- parent(X, Y).` simply defines that being a parent makes one an ancestor. The declarative approach allows programmers to focus on the problem domain, rather than the step-by-step execution process.

The procedural meaning, on the other hand, describes how Prolog searches for an answer. Prolog uses unification, recursion, and backtracking to evaluate queries. When a user asks a question, Prolog follows the execution order of rules, attempting to unify facts and subgoals. If a match is not found, it backtracks to try alternative solutions. This procedural approach determines efficiency and performance, making execution order and rule structure important in complex programs. Thus, while the declarative meaning focuses on what is true, the procedural meaning defines how Prolog derives the answer.
