Experiment No. 1: Family Tree Case Study in PROLOG

To implement a family tree in PROLOG by defining facts, rules, and queries to derive relationships such as parent, ancestor, descendant, grandparent, grandchild, sibling, cousin, uncle, and aunt.

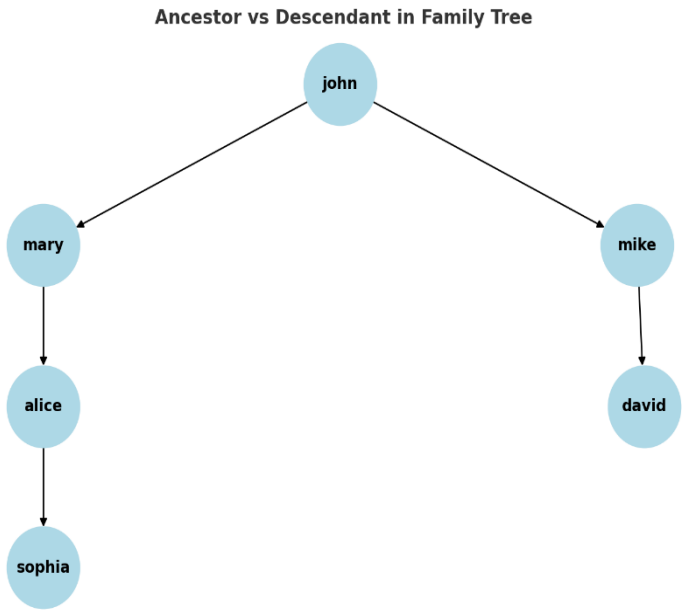
Problem Statement:

In many real-world domains such as genealogy research, healthcare, social networking, and organizational structures, it is necessary to determine relationships among entities. For example, identifying all ancestors of a patient to trace genetic diseases, or retrieving hierarchical relationships in a company for workflow management. Traditional database systems make such recursive queries complex and inefficient.

The problem is to design and implement a knowledge-based family tree system using PROLOG. The system should represent individuals and their parent–child relationships as facts, and automatically infer indirect relationships such as ancestors, descendants, siblings, and cousins using logical rules. It should also allow users to query and retrieve relationship information dynamically.

Algorithm:

1. Define **facts** to represent parent–child relationships.
2. Write **rules** to determine relationships like:
    - Ancestor (direct/indirect parent).
    - Descendant (direct/indirect child).
    - Sibling (sharing the same parent).
    - Grandparent / Grandchild.
    - Uncle / Aunt.
    - Cousins.
3. Execute **queries** to test the system.
4. Display results and verify correctness.



- **Top → Ancestors (like John)**
- **Bottom → Descendants (Mary, Mike, Alice, David, Sophia)**

Program Code (PROLOG)

```
% ---------- FACTS ----------
parent(john, mary).
parent(mary, alice).
parent(john, mike).
parent(mike, david).
parent(alice, sophia).

% ---------- RULES ----------

% Ancestor
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

% Descendant
descendant(X, Y) :- parent(Y, X).
descendant(X, Y) :- parent(Z, X), descendant(Z, Y).
```

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

% Grandparent
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

% Grandchild
grandchild(X, Y) :- grandparent(Y, X).

% Uncle/Aunt
uncle_aunt(X, Y) :- parent(Z, Y), sibling(X, Z).

% Cousin
cousin(X, Y) :- parent(P1, X), parent(P2, Y), sibling(P1, P2).
```

Output:

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['G:/family'].
true.
?- ancestor(john, alice).

true.

?- ancestor(X, sophia).

X = alice ;

X = mary ;

X = john.
```

?- sibling(mike, X).

X = mary.

?- parent(mary, alice).

true.

?- parent(john, X).

X = mary ;

X = mike.

?- parent(alice, X).

X = sophia.

?- sibling(X, Y).

X = mary, Y = mike ;

X = mike, Y = mary.

?- ancestor(mike, david).

true.

?- ancestor(X, david).

X = mike ;

X = john.

?- descendant(X, john).

X = mary ;

X = mike ;

X = alice ;

X = david ;

X = sophia.

?- sibling(alice, mike).

false.

?- sibling(sophia, X).

false.

?- grandparent(X, sophia).

X = mary ;

X = john.

?- grandchild(X, john).

X = alice ;

X = david ;

X = sophia.

?- grandchild(X, mary).

X = sophia.

?- uncle_aunt(X, sophia).

X = mike.

?- cousin(david, X).

X = alice.

?- cousin(alice, david).

true.

Conclusion:

The family tree case study was successfully implemented using PROLOG. The program was able to represent parent–child relationships as **facts** and derive complex relationships like **ancestor, descendant, sibling, grandparent, grandchild, uncle/aunt, and cousin** through logical rules. Queries returned correct results, proving that PROLOG is well-suited for **knowledge representation and reasoning** in Artificial Intelligence applications.