**Fr. Conceicao Rodrigues College of Engineering**

**Department of Computer Engineering**

EXPERIMENT 6

|  |  |
| --- | --- |
| **Practical No:** | 6 |
| **Title:** | **Write a program in prolog to implement for the given facts, Queries and rules.** |
| **Date of Performance:** | 9/4/25 |
| **Date of Submission:** | 27/4/25 |
| **Roll No:** | 9913 |
| **Name of the Student:** | Mark Lopes |

**Rubrics for Evaluation:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Performance Indicator** | **Excellent** | **Good** | **Below Average** | **Total Score** |
| 1 | On time Completion & Submission (01) | 01 (On  Time ) | NA | 00 (Not on Time) |  |
| 2 | Logic/Theory understanding(02) | 02(Correct) | NA | 01 (Tried) |  |
| 3 | Coding Standards (03): Comments/indention/Naming conventions  Output/Test Cases | 03(All used) | 02 (Partial) | 01 (rarely followed) |  |
| 4 | Post Lab Assignment (04) | 04(done well) | 3 (Partially Correct) | 2(submitte d) |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Academic Year** | **2024-25** | **Estimated Time** | **Experiment No. 6 – 02 Hours** |
| **Course & Semester** | **T.E. (CE) – Sem. VI** | **Subject Name** | **CSC604: Artificial Intelligence** |
| **Chapter No.** | **04** | **Chapter Title** | **Knowledge and Reasoning** |
| **Experiment Type** | **Knowledge and Reasoning** | **Software** | **Prolog** |

**AIM: Write a program in prolog to implement for the given facts, Queries and rules.**

**Consider the following facts concerning the Dutch Royal Family.**

**% The facts about the Dutch Royal Family**

mother(wilhelmina,juliana).

mother(juliana,beatrix).

mother(juliana,christina).

mother(juliana,irene).

mother(juliana,margriet).

mother(beatrix,friso).

mother(beatrix,alexander).

mother(beatrix,constantijn).

mother(emma,wilhelmina).

father(hendrik,juliana).

father(bernard,beatrix).

father(bernard,christina).

father(bernard,irene).

father(bernard,margriet).

father(claus,friso).

father(claus,alexander).

father(claus,constantijn).

father(willem,wilhelmina).

queen(beatrix).

queen(juliana).

queen(wilhelmina).

queen(emma).

king(willem).

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

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

ruler(X) :- queen(X).

ruler(X) :- king(X).

Note that these rules have multiple conditions constituting the body that must be satisfied in order for a rule to be satisfied. Also note the indentation of the body in comparison to the head; it makes Prolog programs more readable. The facts and rules above constitute a Prolog program. The clauses of a Prolog program are loaded into a part of the Prolog system called the Prolog database. It can be invoked by means of queries or goals.

The complete Prolog program is included in the file filename.pl. Consult this file, and query the program. If the Prolog system returns with a response, you can enter a semicolon and Prolog will give an alternative solution. If you simply enter <return > Prolog stops looking for alternatives.

**3. Attach the screenshot of the code.**

**Royal\_Dutch\_family**

% Facts

mother(wilhelmina, juliana).

mother(juliana, beatrix).

mother(juliana, christina).

mother(juliana, irene).

mother(juliana, margriet).

mother(beatrix, friso).

mother(beatrix, alexander).

mother(beatrix, constantijn).

mother(emma, wilhelmina).

father(hendrik, juliana).

father(bernard, beatrix).

father(bernard, christina).

father(bernard, irene).

father(bernard, margriet).

father(claus, friso).

father(claus, alexander).

father(claus, constantijn).

father(willem, wilhelmina).

% Monarchs

queen(beatrix).

queen(juliana).

queen(wilhelmina).

queen(emma).

king(willem).

% Rules

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

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

ruler(X) :- queen(X).

ruler(X) :- king(X).

predecessor(X, Y) :-

parent(X, Y),

ruler(X),

ruler(Y).

predecessor(X, Y) :-

ruler(X),

parent(X, Z),

predecessor(Z, Y).

**DATABASE.pl**

% Facts (the database)

employee(mcardon, 1, 5).

employee(treeman, 2, 3).

employee(chapman, 1, 2).

employee(claessen, 4, 1).

employee(petersen, 5, 8).

employee(cohn, 1, 7).

employee(duffy, 1, 9).

department(1, board).

department(2, human\_resources).

department(3, production).

department(4, technical\_services).

department(5, administration).

salary(1, 1000).

salary(2, 1500).

salary(3, 2000).

salary(4, 2500).

salary(5, 3000).

salary(6, 3500).

salary(7, 4000).

salary(8, 4500).

salary(9, 5000).

% -------------------

% Selection Operator

% -------------------

selection(X, Y) :-

call(X), % Check the relation

call(Y), % Check the condition

write(X), % Output the result (tuple)

nl,

fail. % Force backtracking to find all solutions

selection(\_, \_). % End the recursion

% Example Query:

% ?- selection(employee(Name, Department\_N, Scale), (Department\_N = 1, Scale > 2)).

% -------------------

% Projection Operator

% -------------------

projection(X, Y) :-

call(X), % Check the relation

write(Y), % Output the projected attributes

nl,

fail. % Force backtracking to find all solutions

projection(\_, \_). % End the recursion

% Example Query:

% ?- projection(employee(Name, \_, Scale), (Name, Scale)).

% -------------------

% Combine Selection and Projection (sel\_pro)

% -------------------

sel\_pro(X, Y) :-

call(X), % Check the relation

call(Y), % Check the condition

write(Y), % Output the projected attributes

nl,

fail. % Force backtracking to find all solutions

sel\_pro(\_, \_). % End the recursion

% Example Query:

% ?- sel\_pro(employee(Name, Department\_N, Scale), (Department\_N = 1, Scale > 2, (Name, Scale))).

% -------------------

% Join Operator

% -------------------

join(X, Y, Z) :-

call(X), % Check the first relation

call(Y), % Check the second relation

call(Z), % Check the join condition

write(X), % Output the result (joined tuples)

write(Y),

nl,

fail. % Force backtracking to find all solutions

join(\_, \_, \_). % End the recursion

% Example Query:

% ?- join(employee(Name, Department\_N, Scale1),

% salary(Scale2, Amount),

% (Scale1 = Scale2)).

% -------------------

% Join Between employee and department

% -------------------

join\_employee\_department(X, Y, Z) :-

call(X), % Check the employee relation

call(Y), % Check the department relation

call(Z), % Check the join condition

write(X), % Output the result (joined employee and department)

write(Y),

nl,

fail. % Force backtracking to find all solutions

join\_employee\_department(\_, \_, \_). % End the recursion

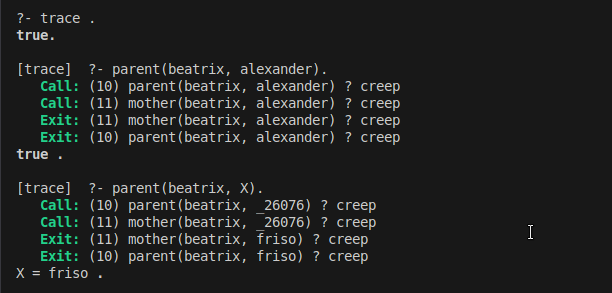
% Example Query:

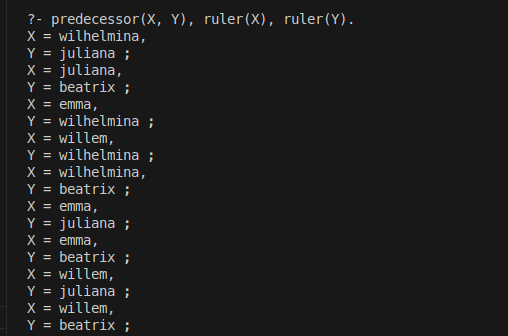
% ?- join\_employee\_department(employee(Name, Department\_N, Scale),

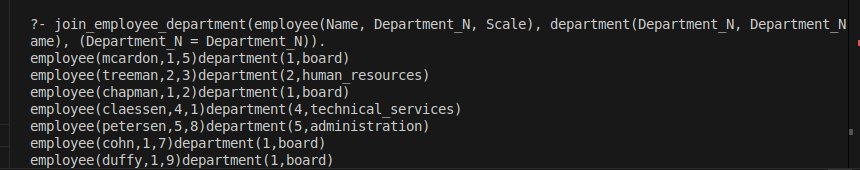
% department(Department\_N, Department\_Name),

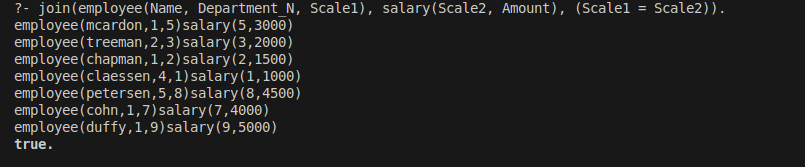
% (Department\_N = Department\_N)).

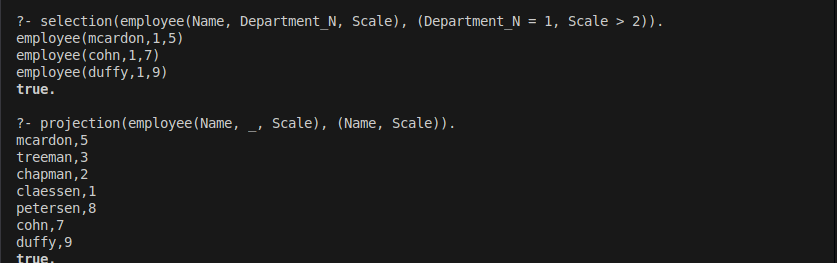
**4. Attach the screenshot of the output.**

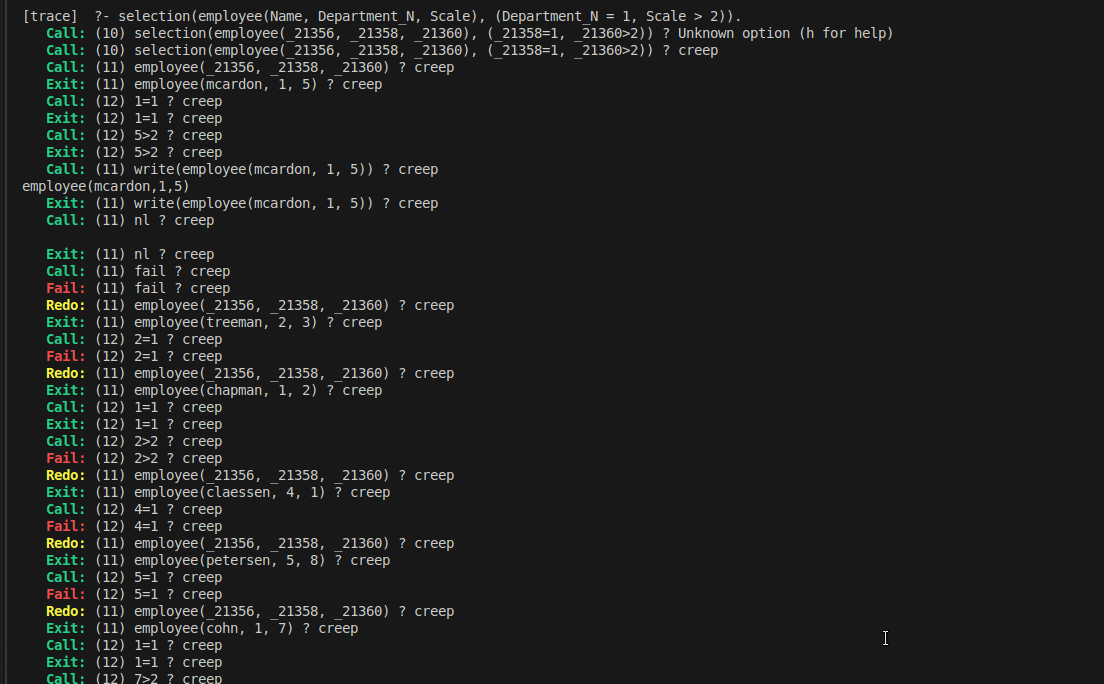












**5. Conclusion**

**In this Prolog example, we’ve demonstrated how to:**

1. **Represent real-world entities (salary scales and amounts) as facts in Prolog.**
2. **Query the database to retrieve specific information such as the salary for a particular scale, all salary scales, or the salary associated with a particular amount.**
3. **Use Prolog's querying mechanism to retrieve values based on known attributes, search for unknowns, and apply logical conditions to find the highest salary.**

**Prolog’s declarative nature allows for elegant and flexible querying over data structures like relational databases.**

**6. Postlab**

Demonstrate how to model and retrieve information from a relational database in a logic-based programming language

1. Represent real-world entities and their relationships in this case, salary(scale, amount) as Prolog facts, where scale and amount are the attributes of the salary relation.
2. Show how to query the database to retrieve specific information based on certain constraints or conditions (e.g., querying the salary for a particular scale, finding all scales, or determining the salary associated with a particular amount).
3. Demonstrate Querying Capabilities: Illustrate how Prolog's querying mechanism works, including how to:

a. Retrieve values based on known attributes (e.g., salary for a given scale).

1. Search for unknowns based on existing facts (e.g., finding a salary scale for a specific salary amount).
2. Use logical conditions (e.g., searching for the highest salary**).**

% Salary facts: salary(Scale, Amount)

salary(a, 50000).

salary(b, 60000).

salary(c, 75000).

salary(d, 85000).

salary(e, 100000).

higher\_salary(Scale1, Scale2) :-

salary(Scale1, Amount1),

salary(Scale2, Amount2),

Amount1 >= Amount2.

