SE2052 – Programming Paradigms

Lab – 07

- What is unification in Prolog, and how is it used to match facts with queries?

  Unification refers to the process in Prolog of making two terms identical by determining the values of any variables within them, and is essentially how Prolog matches a query against facts and rules.

- How does Prolog use backtracking to find multiple solutions to a query?

  If a fact or rule does not completely satisfy the query, then Prolog will automatically attempt a different pathway, or "backtrack" until it finds all possible solutions, or fails altogether.

- What does the cut operator (!)

  The cut operator instructs Prolog in the form of "do not backtrack past this point." The cut operator can be used in a predicate to prevent unnecessary backtracking and therefore attempts to make our programs more efficient or deterministic.

- Explain the Closed World Assumption and how it affects reasoning in Prolog.

  In Prolog, if something is not in the knowledge base, it is assumed to be false. If it can't prove it to be true, Prolog assumes it is false.

- What is the difference between logical negation and negation as failure in Prolog?

  Logical negation*: would indicate true "not true" in classical logic.*
  Negation as failure*: means "Prolog failed to prove Goal." They are not always logical negation.*

- Why does the order of rules and facts matter in Prolog if logic is declarative?

  Though logic is declarative, Prolog executes it in top-down order. Order is important since it handles which answer is found first, and performance.

- Compare backward chaining (used by Prolog) with forward chaining. What are the advantages of each?

  Backward chaining (Prolog): *begins with a query (goal) to find facts/rules. Efficient for particular inquiries.* ***Forward chaining:*** *begins with facts and utilizes rules to apply everything inferred. Better for broad inquiry.*

- How does Prolog handle recursion in rules? Provide an example scenario.

  Prolog uses recursion to solve problems step by step.

- What is the meaning of predicate arity (e.g.,likes/2) and why is it important?

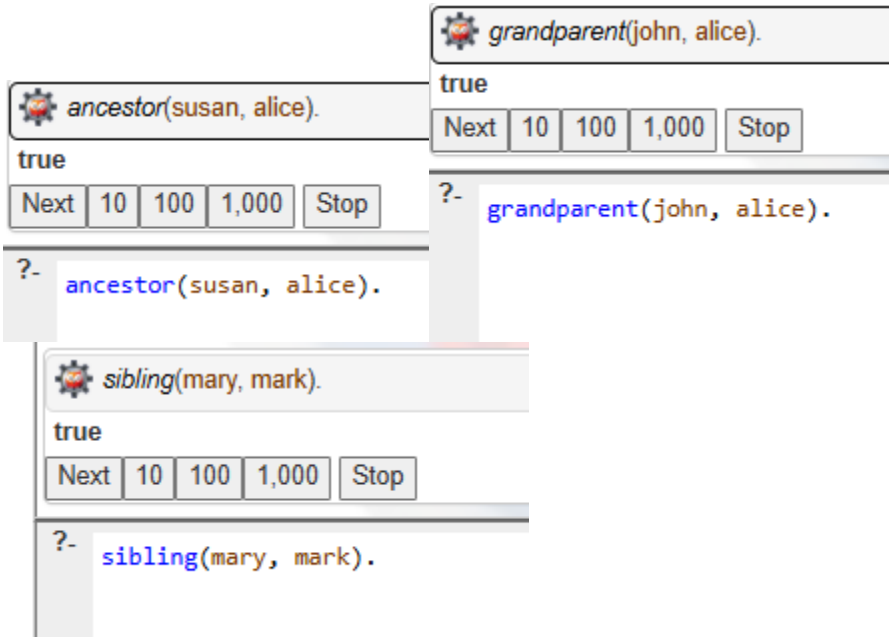
  Arity = number of arguments a predicate takes. likes/2 means likes with 2 arguments. Important because likes/1 and likes/2 are different predicates.

- What are some limitations of Prolog in real-world applications?

  Tends to be slow for very large datasets, Not particularly adept at number crunching or heavy computation, Scaling for complicated systems in the real-world can be difficult, Debugging can be complicated since we assume backtrack we're implemented in the thing being debugged.

Exercise 1:

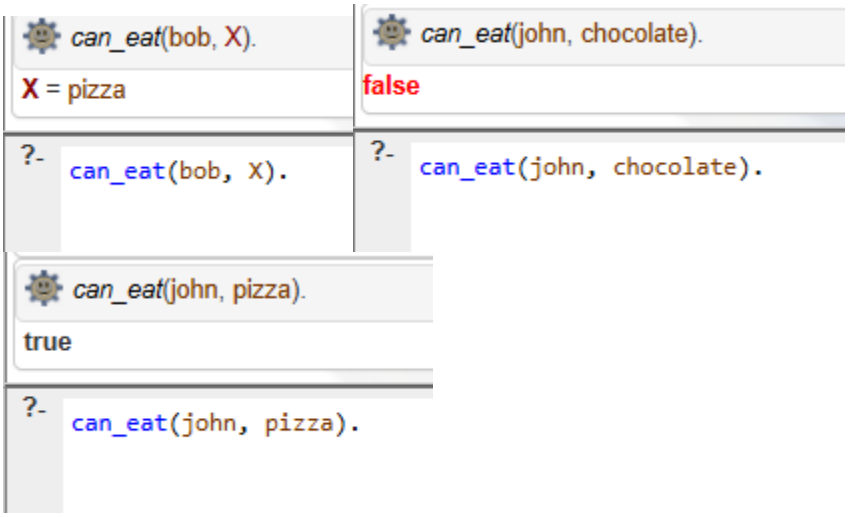
Outputs:



Exercise 2:

```
parent(john, mary).
parent(susan, mary).
parent(john, mark).
parent(susan, mark).
parent(mary, alice).
parent(mark, bob).

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

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

ancestor(X, Y) :-
    parent(X, Y).
ancestor(X, Y) :-
    parent(X, Z),
    ancestor(Z, Y).
likes(john, pizza).
likes(john, salad).
likes(susan, salad).
likes(mary, chocolate).
likes(mark, pizza).
likes(alice, chocolate).
likes(bob, pizza).

dislikes(john, chocolate).
dislikes(susan, pizza).
dislikes(mary, salad).
dislikes(mark, chocolate).
dislikes(alice, pizza).
dislikes(bob, salad).


can_eat(Person, Food) :-
    likes(Person, Food),
    \+ dislikes(Person, Food).
```

Outputs:

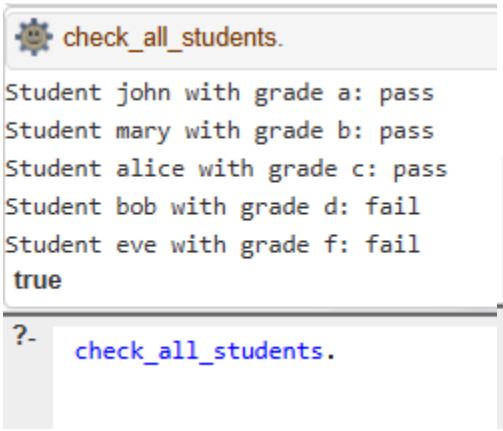


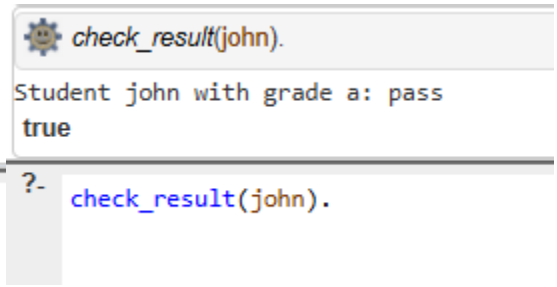
Exercise 3:

Program

```
1
2
3  grade_system(_, a, pass) :-
4      !.
5
6  grade_system(_, b, pass).
7  grade_system(_, c, pass).
8
9  grade_system(_, d, fail).
10 grade_system(_, f, fail).
11
12 student(john, a).
13 student(mary, b).
14 student(alice, c).
15 student(bob, d).
16 student(eve, f).
17
18 check_result(Student) :-
19     student(Student, Grade),
20     grade_system(Student, Grade, Result),
21     format('Student ~w with grade ~w: ~w~n', [Student, Grade, Result]).
22
```

Outputs:





Exercise 4:

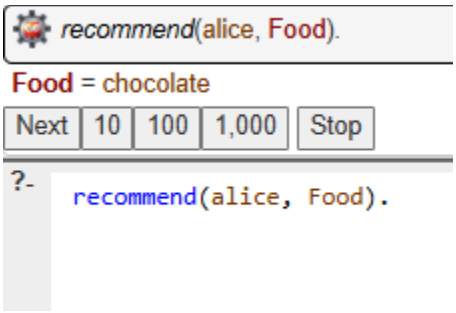
Outputs:


findall(Food, can_eat(john, Food), Foods).
Foods = [pizza, salad]
?- findall(Food, can_eat(john, Food), Foods).
likes(john, Food).
Food = pizza
Next 10 100 1,000 Stop
?- likes(john, Food).

Exercise 5:


SWISH
File
Edit
Examples
Program
parent(john, mary).
parent(susan, mary).
parent(john, mark).
parent(susan, mark).
parent(mary, alice).
parent(mark, bob).
sibling(X, Y) :-
parent(P, X),
parent(P, Y),
X \= Y.
grandparent(X, Y) :-
parent(X, Z),
parent(Z, Y).
ancestor(X, Y) :-
parent(X, Y).
ancestor(X, Y) :-
parent(X, Z),
ancestor(Z, Y).
likes(john, pizza).
likes(john, salad).
likes(john, chocolate).
likes(susan, salad).
likes(susan, chocolate).
likes(mary, chocolate).
likes(mary, pizza).
likes(mark, pizza).
likes(mark, salad).
likes(alice, chocolate).
likes(alice, fruit).
likes(bob, pizza).
likes(bob, salad).
dislikes(john, chocolate).
dislikes(susan, pizza).
dislikes(mary, salad).
dislikes(mark, chocolate).
dislikes(alice, pizza).
dislikes(bob, salad).
can_eat(Person, Food) :-
likes(Person, Food),
\+ dislikes(Person, Food).
recommend(Person, Food) :-
likes(Person, Food),
\+ dislikes(Person, Food).

Output:


recommend(alice, Food).
Food = chocolate
Next
10
100
1,000
Stop
?-
recommend(alice, Food).