

# DEPARTMENT OF INFORMATION TECHNOLOGY

|  |  |
| --- | --- |
| **COURSE CODE: DJ22ITL501** | **DATE:** |
| **COURSE NAME: Artificial Intelligence Laboratory** | **CLASS: TY-IT** |

**EXPERIMENT NO.08**

**CO/LO:** Formulate the problem as a state space and select appropriate technique from blind, heuristic or adversarial search to generate the solution.

**AIM / OBJECTIVE:** Implement a suitable case study using PROLOG.

# DESCRIPTION OF EXPERIMENT:

Prolog is to define a set of rules that describe the relationships between different objects or concepts in your problem domain. For example, you might define rules that specify that certain objects are bigger than others, or that some objects are the same color. Then, you can use Prolog to ask questions about these objects and their relationships, and the interpreter will use your rules to deduce the answers.

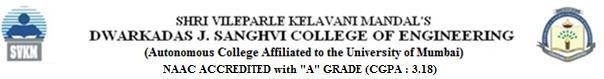
To use Prolog, you will need to have a Prolog interpreter installed on your computer. There are several different Prolog interpreters available, including [SWI-Prolog](https://www.swi-prolog.org/)[, GNU Prolog](https://www.gprolog.org/) and [B-Prolog.](https://www.picat-lang.org/bprolog/) Once you’ve installed an interpreter, you can start writing Prolog programs using a [text editor](https://builtin.com/software-engineering-perspectives/code-editors) and then run them using the interpreter.

Prolog is a powerful and flexible programming language that’s well-suited for developing logicbased [artificial intelligence](https://builtin.com/artificial-intelligence) applications. It allows the [programmer](https://builtin.com/learn/careers/software-engineer) to specify a set of rules and facts about a problem domain, and then use those rules and facts to automatically infer solutions to problems.

There is no single “syntax” for Prolog, as the language allows for a wide range of different programming styles and approaches. However, here are some basic elements of Prolog syntax that are commonly used:

* Facts are statements that are assumed to be true. In Prolog, facts are written using a predicate name followed by a list of arguments enclosed in parentheses. For example: man(john).
* Rules are logical statements that describe the relationships between different facts. In Prolog, rules are written using the predicate name followed by a list of arguments enclosed in parentheses, followed by a colon and a hyphen (:-) and the body of the rule.

For example: not(X,Y) :- man(X), woman(Y).



* Variables are used to represent values that can change or be determined by the interpreter. In Prolog, variables are written using a name that begins with an uppercase letter. For example: X.
* Queries are used to ask the interpreter to find solutions to problems based on the rules and facts in the program. In Prolog, queries are written using the same syntax as facts followed by a question mark (?). For example: not(john, mary)?

# EXPLANATION / SOLUTIONS (DESIGN):

**Code:**

simple Prolog program might look like this:

man(john). woman(mary).

capital\_of(france, paris).

not(X,Y) :- man(X), woman(Y).

# Output:

The first three lines are facts, while the fourth line is a rule. The rule uses the not/2 predicate to state that if X is a man and Y is a woman, then X is not Y.

# Example:

**Example 1:** Below food table shows the facts, rules, goals and their english meanings.

# English

|  |  |
| --- | --- |
| **Facts** | **meanings** |
| food(burger). | // burger is a food |
| food(sandwich). | // sandwich is a food |
| food(pizza). | // pizza is a food |
| lunch(sandwich). | // sandwich is a lunch |
| dinner(pizza). | // pizza is a dinner |
| Rules |  |
| meal(X) :- food(X). | // Every food is a meal OR  Anything is a meal if it is  a food |
| **Queries / Goals** |  |
| ?- food(pizza). | // Is pizza a food? |
| ?- meal(X), lunch(X). | // Which food is meal and lunch? |

|  |  |
| --- | --- |
| ?- dinner(sandwich). | // Is sandwich a dinner? |

**Example 2 : Below student-professor relation table shows the facts, rules, goals and their english meanings**.

|  |  |
| --- | --- |
| **Facts** | **English meanings** |
| studies(charlie, csc135). | // charlie studies csc135 |
| studies(olivia, csc135). | // olivia studies csc135 |
| studies(jack, csc131). | // jack studies csc131 |
| studies(arthur, csc134). | // arthur studies csc134 |
| teaches(kirke, csc135). | // kirke teaches csc135 |
| teaches(collins, csc131). | // collins teaches csc131 |
| teaches(collins, csc171). teaches(juniper, csc134).  **Rules** professor(X, Y) :-  teaches(X, C), studies(Y, C).  **Queries / Goals**  ?- studies(charlie, What).  ?- professor(kirke, Students). | // collins teaches csc171  // juniper teaches csc134  // X is a professor of Y if X teaches C and Y studies C.  // charlie studies what? OR What does charlie study?  // Who are the students of professor kirke. |

**Example 4:** holi. cat(bengal). /\* bengal is a cat \*/ dog(rottweiler). /\* rottweiler is a dog \*/ likes(Jolie, Kevin). /\* Jolie likes Kevin \*/ likes(A, Kevin). /\* Everyone

likes Kevin \*/ likes(Jolie, B). /\* Jolie likes everybody \*/

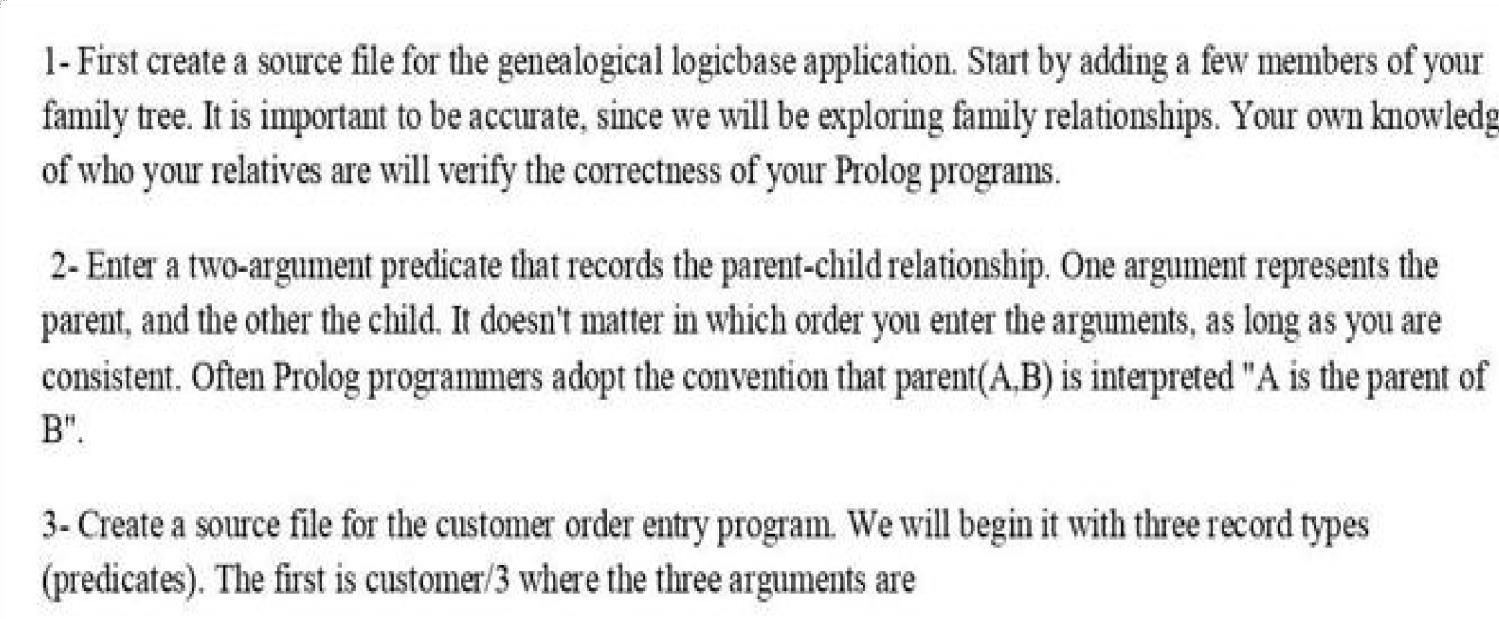
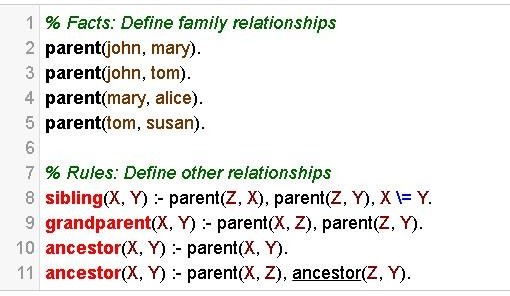
likes(B, Jolie), likes(Jolie, B). /\* Everybody likes Jolie and Jolie likes everybody \*/ likes(Jolie, Kevin); likes(Jolie, Ray). /\* Jolie likes Kevin or Jolie likes Ray \*/ not(likes(Jolie, pasta)). /\* Jolie does not like pasta \*/ **Output:**

?- 'It is sunny'. yes

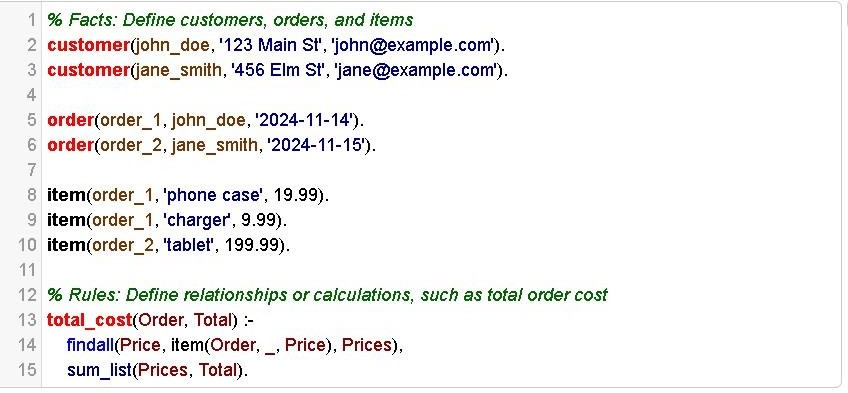
?- 'It is cold'. no

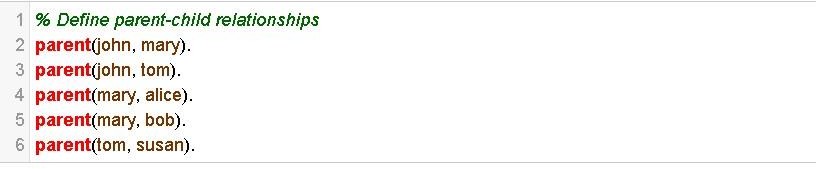
?-

# Program Design: Code:

**1)**

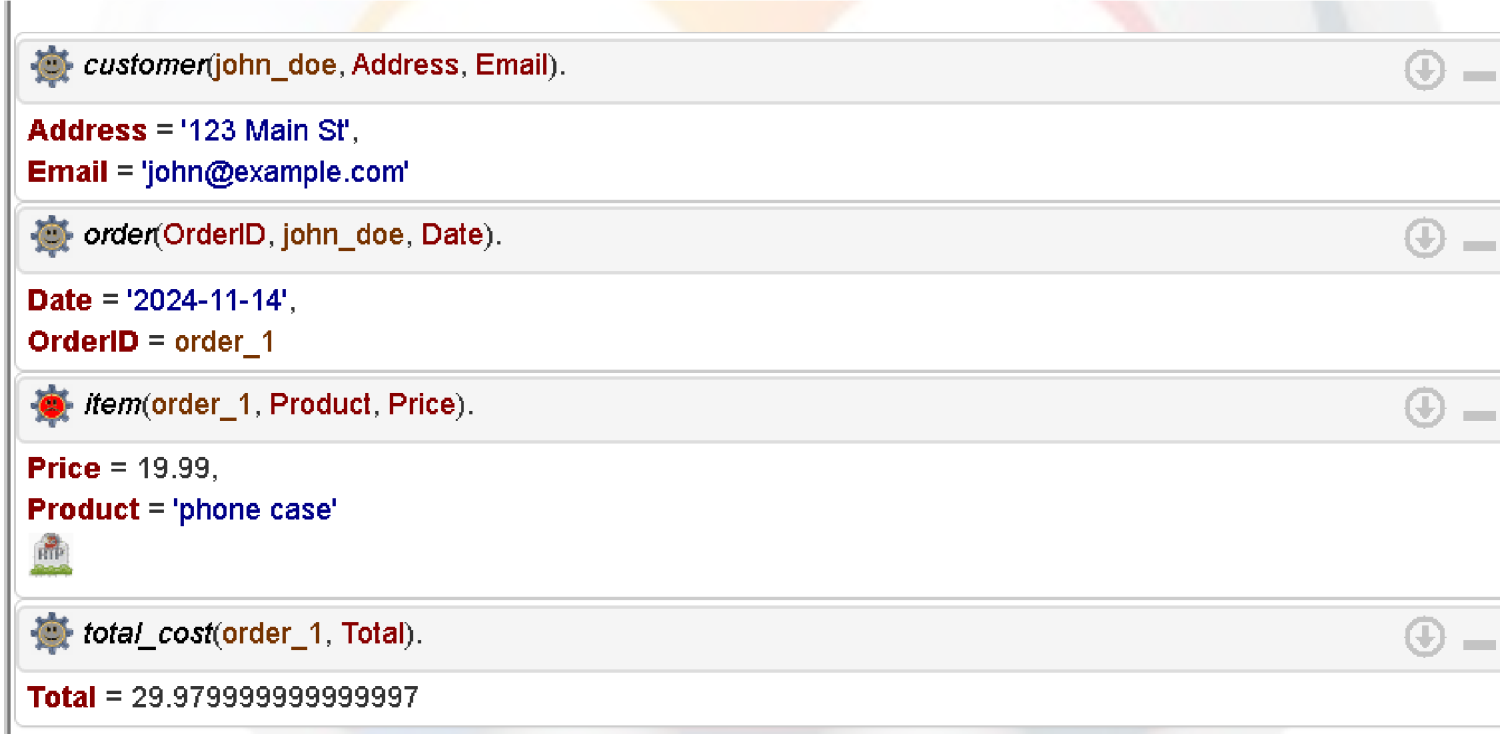
# 2)



**3)**

# Output:

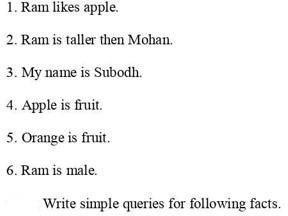
1)



2)

3)

# Questions:



1. likes(ram, apple).
2. taller\_than(ram, mohan). 3)name(myself, subodh). 4)fruit(apple).
3. fruit(orange). 6)male(ram).

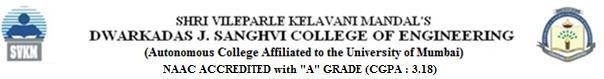
**CONCLUSION:** We implemented facts and rules using prolog .

# REFERENCES:

* 1. Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition,

Pearson Education, 2010

* 1. <https://silp.iiita.ac.in/wp-content/uploads/PROLOG.pdf>



* 1. https://swish.swi-prolog.org/