TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
LAB REPORT
ON

PROLOG - I : BASICS OF PROLOG

SUBMITTED BY:

Diwas Adhikari **(076BEI014)**

SUBMITTED TO:

Department of Electronics & Computer Engineering
Pulchowk Campus, IOE

LAB NUMBER

1

SUBMISSION DATE

July 12, 2023

(For course completion of BEI – Artificial Intelligence)

OBJECTIVES

- To understand fundamentals of PROLOG and construct logical equivalences using its programming paradigms and techniques.
- To solve simple problems in PROLOG through its logic-based approach using facts and rules (clauses).

THEORY

PROLOG is a programming language that is well-suited for developing logic-based artificial intelligence applications. It is a declarative programming language which suggests that it allows the programmer to specify the rules and facts about a problem domain, and then the PROLOG interpreter will use these rules and facts to automatically infer solutions to problems.

PROLOG is a powerful and flexible programming language which was created by Alain Colmerauer, Phillipe Roussell and Robert Kowalski in 1972 as an alternative to the American-dominated LISP programming language. It is an attempt to make a programming language that enables the expression of logic instead of carefully specified instructions on the computer. There are several different interpreters available such as SWI-PROLOG, GNU-PROLOG, and B-PROLOG for its programming.

Data Types in PROLOG

- <u>Atoms and numbers</u>
  Atoms can be constructed in three different ways:-
  1. Strings of letters, digits, and the underscore character '_' starting with a lower case letter. For example: harry, harry_potter, harry_potter_07, etc.
  2. Strings of characters enclosed in quotes. For example: 'Harry', 'Potter', etc.
  3. Strings of special characters. For example: <------>, ::::::::, etc.
     [Any use of character combination associated to an in-built feature should be avoided.]

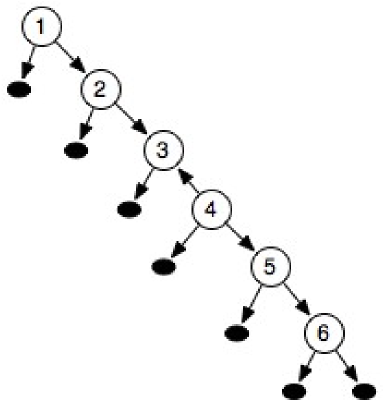
  Numbers used in PROLOG are integers and real numbers itself.

- <u>Variables</u>
  Variables are strings of letters, digits and underscore that start with an underscore or an upper-case letter. The scope of a variable is limited to a single clause. So, the same variable used in different clauses associated to a different value.
  For example: X, Sum, _count etc. If a variable is to be not inferred more than once in a clause, an underscore '_' is used to make an anonymous variable.

- <u>Structures</u>
  Structures are objects that have different components. The components can be atoms or yet some other structures. A functor is used to construct a structure.

For example: Egypt(Pharaoh, Queen, Servants) where Egypt is a structure that has Pharaoh, Queen and Servants as its elements. Pharaoh and Queen may be atoms while Servants may be yet another structure or a list of atoms.

List is a special built in structure in PROLOG. It can be represented as a sequence of elements ordered linearly. However, it is internally represented as a binary tree.



For example: if _pantheon is a list of atoms (in this case - strings), its elements are: [‘Ra’ , ‘Isis’, ‘Horus’, ‘Set’]. The list can be decomposed into two parts: HEAD and TAIL where HEAD (H) is the first element of the list and TAIL (T) is the remaining list. The above list can be broken down as [H| T].

where H= ‘Ra’
and T= [‘Isis’, ‘Horus’, ‘Set’]

Writing Programs & Execution in PROLOG

All PROLOG programs start from the goal. Then, it uses the facts and clauses to break down the goal to sub-goals and tries to prove the subgoals. A clause is said to have succeeded if there is a combination of facts and clauses that holds true. PROLOG has built in backtracking mechanism such that even if a rule declares a query to be false, it goes through all the possible paths for verifying the goal to finally declare it false if the solution can’t be devised out of those rules and clauses. This strategy is simply called chronological backtracking.

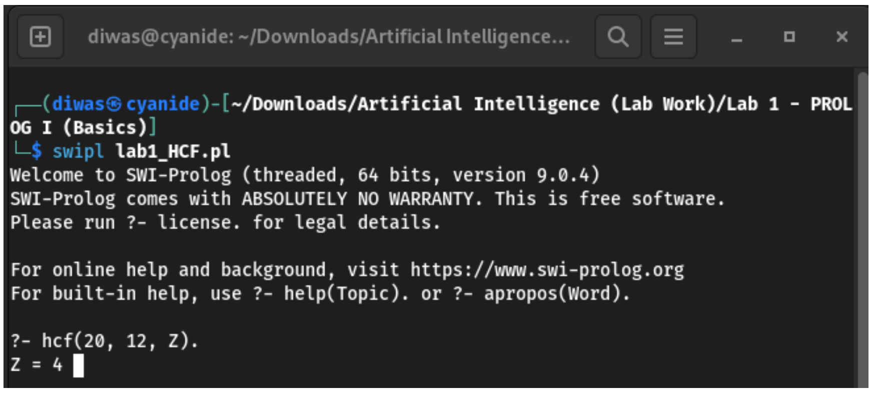
Features of PROLOG

1. Unification: Multiple phrases can be defined to represent the same structure.
2. Automated backtracking: PROLOG traces backwards to process every logical path.
3. Recursion: Recursion in PROLOG handles searching in trees which uses the facts and clauses to break that makes it a formidable inference engine.
4. Parallelism: PROLOG supports (speculative) parallelism in a context where some computations can be performed early for its use in the future.

PROLOG CODE & OBSERVATION

1) Evaluate HCF of two numbers.

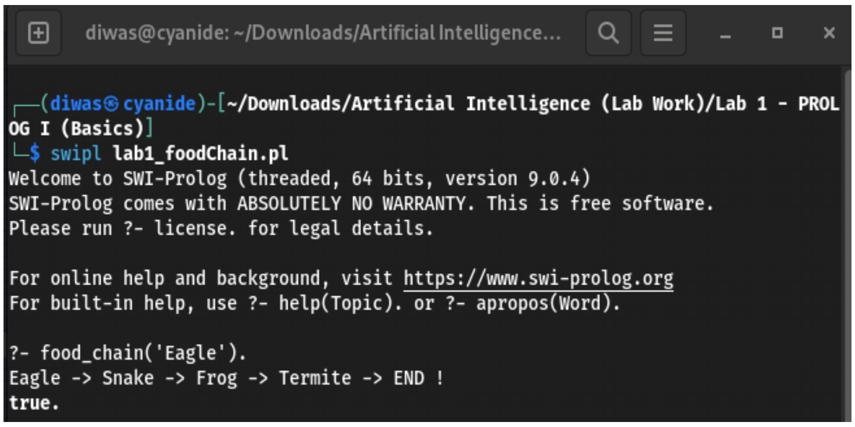
```
hcf(X, Y, Z) :- X=:=Y, Z is X ;
        X>Y, hcf(X-Y, Y, Z) ;
        X<Y, hcf(X, Y-X, Z).
```



```
diwas@cyanide: ~/Downloads/Artificial Intelligence...

┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_HCF.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- hcf(20, 12, Z).
Z = 4
```

2) Program of choice – Food Chain

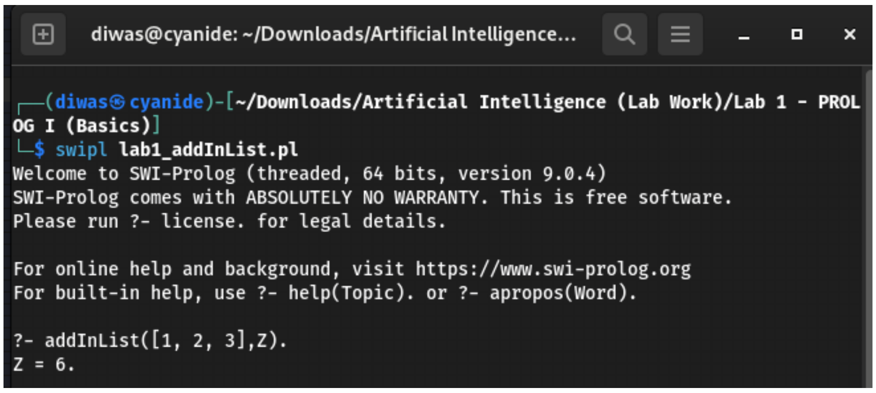
```
kills('Termite', '').
kills('Frog', 'Termite').
kills('Snake', 'Frog').
kills('Eagle', 'Snake').

food_chain(A):-
        write(A),
        kills(A, B), write(' -> '),
        food_chain(B).

food_chain(''):-
        write('END !').
```



```
diwas@cyanide: ~/Downloads/Artificial Intelligence...

┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_foodChain.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- food_chain('Eagle').
Eagle -> Snake -> Frog -> Termite -> END !
true.
```

3) Add the content of an integer list and display its result

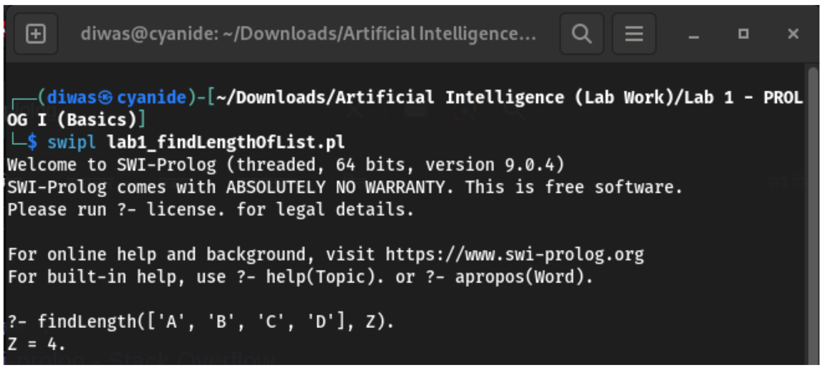
```
addInList([], Sum):-
        Sum is 0.
addInList([H|T], Sum) :-
        addInList(T, Prev_sum),
        Sum is H + Prev_sum.
```



```
diwas@cyanide: ~/Downloads/Artificial Intelligence...

┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_addInList.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- addInList([1, 2, 3],Z).
Z = 6.
```

4) Find the length of a list.

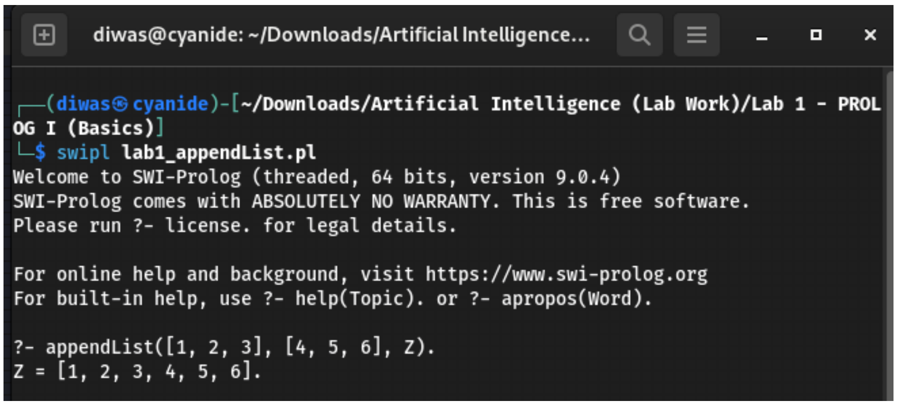
```
findLength([], Length):-
        Length is 0.
findLength([H|T], Length):-
        findLength(T, Count),
        Length is Count + 1.
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_findLengthOfList.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- findLength(['A', 'B', 'C', 'D'], Z).
Z = 4.
```

5) Append two lists.

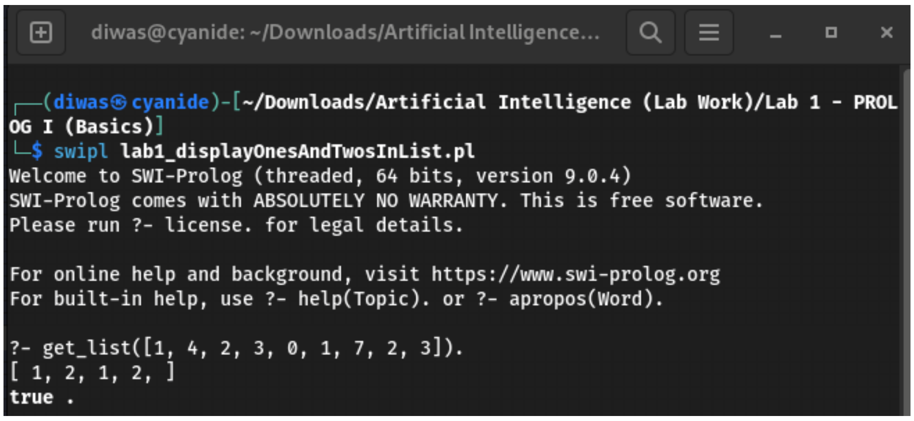
```
appendList([], My_list, My_list).
appendList([H|T], My_list, [H|_temp]):-
        appendList(T, My_list, _temp).
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_appendList.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- appendList([1, 2, 3], [4, 5, 6], Z).
Z = [1, 2, 3, 4, 5, 6].
```

6) Take a list of integers and display only 1's and 2's from the list.

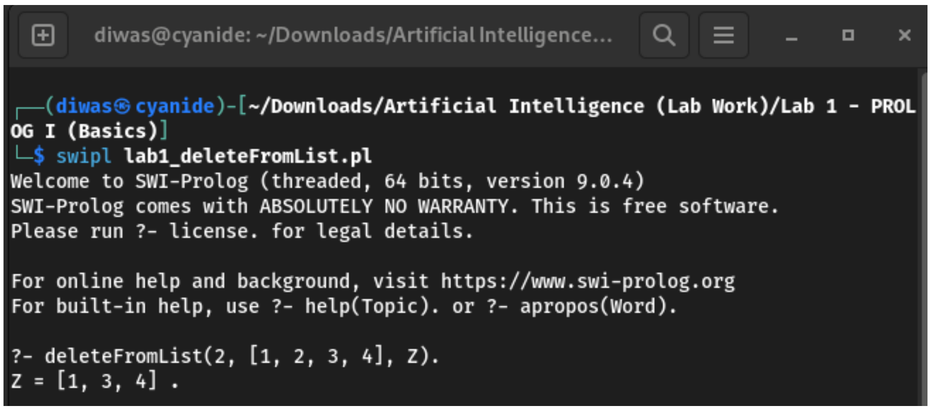
```
get_list(My_list):-
        write("[ "),
        displayOnesAndTwos(My_list).
displayOnesAndTwos([H|T]):-
        (H =:= 1 ; H =:= 2), % Only write the values if it is equal to 1 or 2.
        write(H),
        write(', '),
        displayOnesAndTwos(T).
displayOnesAndTwos([H|T]):-
        displayOnesAndTwos(T).
displayOnesAndTwos([]):-
        write("]").
```



```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_displayOnesAndTwosInList.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- get_list([1, 4, 2, 3, 0, 1, 7, 2, 3]).
[ 1, 2, 1, 2, ]
true .
```

7) Delete a given element in a list.

```
% If list has only that item, show null list.
deleteFromList(Item, [Item], []).
% If list is itself at the head, just show tail list.
deleteFromList(Item, [Item|My_list], My_list).
% Else find item in the list and skip that item on final list.
deleteFromList(Item, [H|My_list], [H|Temp_list]):-
        deleteFromList(Item, My_list, Temp_list).
```



diwas@cyanide: ~/Downloads/Artificial Intelligence...

```
┌──(diwas㉿cyanide)-[~/Downloads/Artificial Intelligence (Lab Work)/Lab 1 - PROL
OG I (Basics)]
└─$ swipl lab1_deleteFromList.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- deleteFromList(2, [1, 2, 3, 4], Z).
Z = [1, 3, 4] .
```

DISCUSSION & CONCLUSION

PROLOG turned out to be a very powerful language which has a different programming paradigm than other langugages. But, still PROLOG is a very concise language which is rigid enough for efficient inference. We encountered warnings of singleton variables which indicates that there is one or more variable in the clause that appears only once. It was suppressed by using the following in-built clause.

```
:-style_check(-singleton).
```

Hence, in this way, we understood the basics of PROLOG and used it to perform simple computation using logic-based approach using rules and clauses.

SUBMITTED BY:

Diwas Adhikari
076BEI014 – (Roll Number: 14)
Bachelor in Electronics & Communication Engineering (BEI)
076bei014.diwas@pcampus.edu.np
lunaticoda123@gmail.com