UNIVERSITY OF DELHI

ATMA RAM SANATAN DHARMA COLLEGE

(Discipline Specific Core **DSCCS02***)*

Artificial Intelligence

PRACTICAL FILE

Submitted To:

MS. RASHMI YADAV

Submitted By:

Divyansh Joshi
24/48057
B.Sc. (H) Computer Science
Examination Rollno : 24003570035

S.no.	Practical	Date
01	Write a PROLOG program to implement the family tree and demonstrate the family relationship.	Aug 10
02	Write a PROLOG program to implement conc(L1, L2, L3) where L2 is the list to be appended with L1 to get the resulted list L3.	Aug 14
03	Write a PROLOG program to implement reverse(L, R) where List L is original and List R is reversed list.	Aug 21
04	Write a PROLOG program to calculate the sum of two numbers.	Aug 21
05	Write a PROLOG program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.	Aug 28
06	Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.	Aug 28
07	Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the Fibonacci series.	Sep 11
08	Write a PROLOG program to implement power (Num, Pow, Ans) : where Num is raised to the power Pow to get Ans.	Sep 11
09	PROLOG program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.	Oct 09
10	Write a PROLOG program to implement memb(X, L): to check whether X is a member of L or not.	Oct 10
11	Write a PROLOG program to implement sumlist(L, S) so that S is the sum of a given list L.	Oct 30
12	Write a PROLOG program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.	Oct 30
13	Write a PROLOG program to implement maxlist(L, M) so that M is the maximum number in the list.	Nov 06
14	Write a PROLOG program to implement insert(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.	Nov 06
15	Write a PROLOG program to implement delete(N, L, R) that removes the element on Nth position from a list L to generate a list R.	Nov 06

Question

1.Write a Prolog program to implement the family tree and demonstrate the family relationship.

Program:

```
% Genders
male(dashrath).
male(rama).
male(lakshmana).
male(bharata).
male(shatrughna).
male(luv).
male(kusha).
male(mukul).
male(ayush).
male(laksh).

female(kaushalya).
female(sumitra).
female(kaikeyi).
female(sita).
female(urmila).
female(mandavi).
female(shrutakirti).
female(sneha).
female(karishma).
female(siddhi).
female(diya).
female(kanishka).

% Parent-child relationships: parent(Parent, Child)
parent(dashrath, rama).
parent(dashrath, lakshmana).
parent(dashrath, bharata).
parent(dashrath, shatrughna).
parent(kaushalya, rama).
parent(sumitra, lakshmana).
parent(sumitra, shatrughna).
parent(kaikeyi, bharata).

parent(rama, luv).
parent(rama, kusha).
parent(sita, luv).
parent(sita, kusha).

% 'My Family'
parent(mukul, ayush).
```

```
% 'My Family'
parent(mukul, ayush).
parent(mukul, karishma).
parent(sneha, ayush).
parent(sneha, karishma).

parent(ayush, diya).
parent(ayush, kanishka).
parent(siddhi, diya).
parent(siddhi, kanishka).

parent(karishma, laksh).


% Marriages: married(Spouse1, Spouse2) - male,female order
married(dashrath, kaushalya).
married(dashrath, sumitra).
married(dashrath, kaikeyi).
married(rama, sita).
married(lakshmana, urmila).
married(bharata, mandavi).
married(shatrughna, shrutakirti).
married(mukul, sneha).
married(ayush, siddhi).


%Defining derived relationships

% Child of someone
child(Child, Parent) :- parent(Parent, Child).

% Father and Mother
father(Father, Child) :- male(Father), parent(Father, Child).
mother(Mother, Child) :- female(Mother), parent(Mother, Child).

% Son and Daughter
son(Son, Parent) :- male(Son), parent(Parent, Son).
daughter(Daughter, Parent) :- female(Daughter), parent(Daughter, Parent).

% Grandparent and Grandchild
grandparent(Grandparent, Grandchild) :-
```

```
% Grandparent and Grandchild
grandparent(Grandparent, Grandchild) :-
    parent(Grandparent, Parent),
    parent(Parent, Grandchild).

grandfather(Grandfather, Grandchild) :-
    male(Grandfather),
    grandparent(Grandfather, Grandchild).

grandmother(Grandmother, Grandchild) :-
    female(Grandmother),
    grandparent(Grandmother, Grandchild).

grandchild(Grandchild, Grandparent) :-
    grandparent(Grandparent, Grandchild).

% Siblings
sibling(Person1, Person2) :-
    parent(P, Person1),
    parent(P, Person2),
    Person1 \= Person2. % person 1 and person 2 not same

brother(Brother, Person) :-
    male(Brother),
    sibling(Brother, Person).

sister(Sister, Person) :-
    female(Sister),
    sibling(Sister, Person).
 %other relations -uncle, aunt, cousins
uncle(Uncle, NieceNephew) :-
    male(Uncle),
    sibling(Uncle, ParentOfNieceNephew),
    parent(ParentOfNieceNephew, NieceNephew).

uncle(Uncle, NieceNephew) :-
    male(Uncle),
    married(Uncle, SiblingOfParent),
    sibling(SiblingOfParent, ParentOfNieceNephew),
    parent(ParentOfNieceNephew, NieceNephew).
```

```
aunt(Aunt, NieceNephew) :-
    female(Aunt),
    sibling(Aunt, ParentOfNieceNephew),
    parent(ParentOfNieceNephew, NieceNephew).

aunt(Aunt, NieceNephew) :-
    female(Aunt),
    married(Aunt, SiblingOfParent),
    sibling(SiblingOfParent, ParentOfNieceNephew),
    parent(ParentOfNieceNephew, NieceNephew).

cousin(Person1, Person2) :-
    parent(P1, Person1),
    parent(P2, Person2),
    sibling(P1, P2),
    Person1 \= Person2.
```

Output:

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

?- [familytree].
true.

?- brother(Bro, rama).
Bro = lakshmana ;
Bro = bharata ;
Bro = shatrughna .

?- grandchild(GC, dashrath).
GC = luv ;
GC = kusha .

?- father(F, ayush).
F = mukul .

?- aunt(A, diya).
A = karishma .

?- sibling(luv, kusha).
true
```

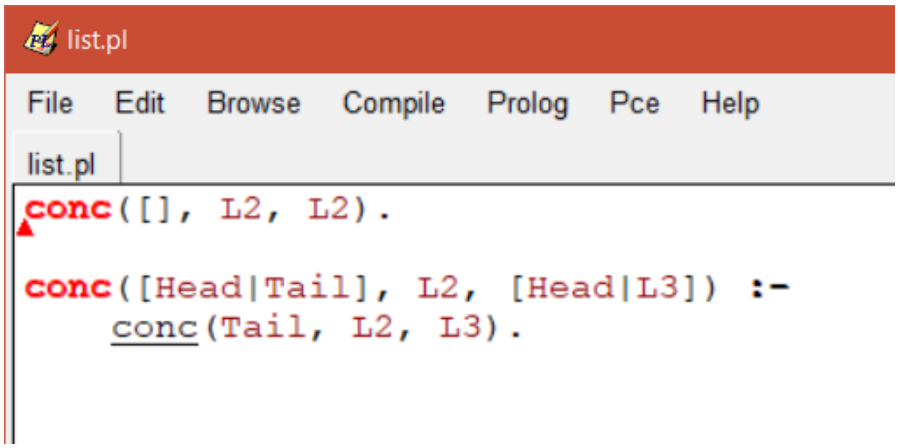
Learning Outcome

- I understood how PROLOG represents family relationships using simple facts like parent and gender.
- I learned how rules help PROLOG figure out relationships such as siblings or grandparents.
- I saw how PROLOG uses pattern matching to connect different pieces of information.
- I realized that PROLOG thinks in terms of “truths” instead of step-by-step instructions.
- I now understand how a family tree can be built logically using facts and rules instead of diagrams.

Question

2.Write a Prolog program to implement conc(L1, L2, L3) where L2 is the first to be appended with L1to get resulted list L3.

Program:



Output:

```
?- [list].
true.

?- conc([a, b], [c, d], L3).
L3 = [a, b, c, d].

?- conc([1, 2], [3], [1, 2, 3]).
true.

?- conc(L1, L2, [apple, orange, banana]).
L1 = [],
L2 = [apple, orange, banana] ;
L1 = [apple],
L2 = [orange, banana] ;
L1 = [apple, orange],
L2 = [banana] ;
L1 = [apple, orange, banana],
L2 = [] ;
false.

?- conc([a, b], L2, [a, b, c, d]).
L2 = [c, d].

?-
```

Learning Outcome

- I learned how PROLOG breaks lists into head and tail to join them together.
- I understood how recursion helps build the final list one element at a time.
- I saw how the base case stops the combining process when the first list ends.
- I realized that PROLOG builds new lists by attaching elements to the front.
- I now understand how list operations work in PROLOG using simple patterns instead of loops.

Question

3.Write a PROLOG program to implement reverse(L, R) where List L is original and List R is reversed list.

Program:

reversal.pl

File Edit Browse Compile Prolog Pce Help

reversal.pl

```
reverse_list(List, ReversedList) :-
    reverse_helper(List, [], ReversedList).

reverse_helper([], ReversedList, ReversedList).

reverse_helper([H|T], Acc, ReversedList) :-
    reverse_helper(T, [H|Acc], ReversedList).
```

Output:

SWI-Prolog (Multi-threaded, version 7.2.3)

File Edit Settings Run Debug Help

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 7.2.3)
Copyright (c) 1990-2015 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- [reversal].
true.

2 ?- reverse_list([1, 2, 3, 4], Reversed).
Reversed = [4, 3, 2, 1].

3 ?- reverse_list([1, 2], Reversed).
Reversed = [2, 1].

4 ?- reverse_list([], Reversed).
Reversed = [].

5 ?- reverse_list([1], Reversed).
Reversed = [1].

6 ?-
```

Learning Outcome

- I learned how reversing a list depends on taking the first element and placing it at the end.
- I understood how recursion processes each item until the list becomes empty.
- I saw how the base case (empty list) gives PROLOG a stopping point.
- I realized that the append step is what rebuilds the reversed list.
- I now understand how PROLOG handles list transformations using repeated smaller steps.

Question

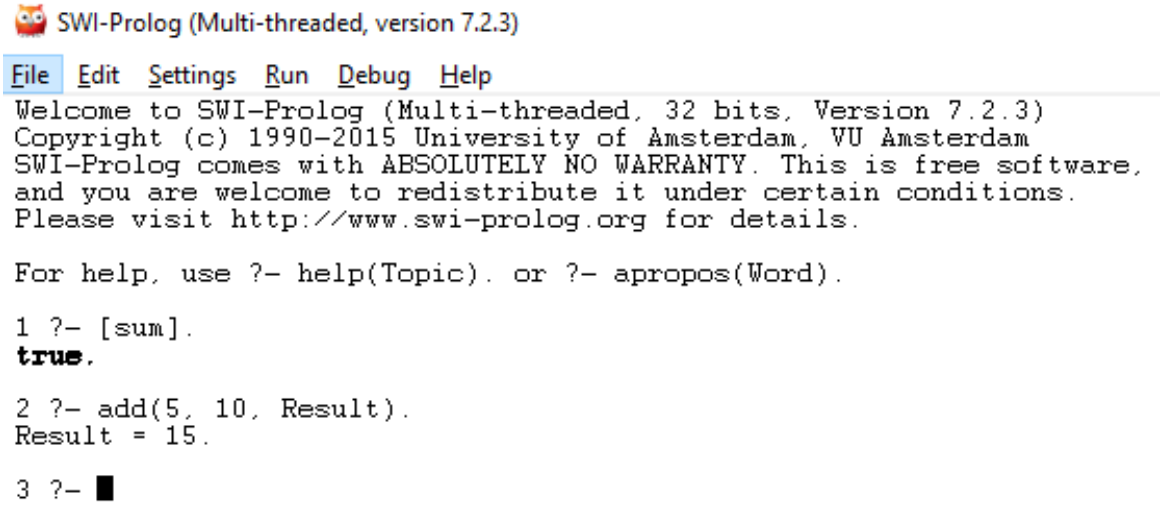
4.Write a PROLOG program to calculate the sum of two numbers.

Program:

sum.pl

```
add(Num1, Num2, Sum) :-
    Sum is Num1 + Num2.
```

Output:



Learning Outcome

- I learned how PROLOG uses the “is” operator to calculate arithmetic results.
- I understood that PROLOG doesn't compute automatically unless told to evaluate expressions.
- I saw how numbers can be passed as arguments and processed inside rules.
- I realized that even simple math operations need logical statements in PROLOG.
- I now understand how PROLOG handles basic calculations using clear and direct rules.

Question

5.Write a PROLOG program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.

Program:

maxmin.pl

```
%If X is greater than or equal to Y, then X is the maximum.
max(X, Y, X) :- X >= Y.

%If Y is greater than X, then Y is the maximum.
max(X, Y, Y) :- X < Y.
```

Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File  Edit  Settings  Run  Debug  Help
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).

?- [maxmin].
true.

?- max(5, 10, M).
M = 10.

?- max(20, 15, M).
M = 20
```

Learning Outcome

I learned that PROLOG doesn't really have an "if/else" statement like other programming languages. Instead, you just write a few different rules for the same problem. For the max program, I wrote one rule that says, "if X is bigger, then X is the answer," and another rule that says, "if Y is bigger, then Y is the answer." It's like giving the program a few different paths to follow, and it automatically picks the right one based on the numbers you give it.

Question

6.Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.

Program:

factorial.pl

```
factorial(0, 1).
factorial(N, F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, F1),
    F is N * F1.
```

Output:

```
File  Edit  Settings  Run  Debug  Help
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).

?- [factorial].
true.

?- factorial(5, F).
F = 120 ;
false.

?- factorial(3, F).
F = 6 ;
false.

?-
```

Learning Outcomes

Recursion just means a function that calls itself. To make it work, you absolutely need two parts:

- The end-of-the-road rule: This is the factorial(0, 1). part. It's the only rule that doesn't call itself, so it's what stops the program from running forever.

- The keep-going rule: This is the part that tells the program, "to find the factorial of N, first find the factorial of N minus 1." It keeps doing this over and over until it hits that first rule.

Question

7.Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the Fibonacci series.

Program:

fibonacci.pl

```
fib(0, 0).
fib(1, 1).
fib(N, F) :-
    N > 1,
    N1 is N - 1,
    N2 is N - 2,
    fib(N1, F1),
    fib(N2, F2),
    F is F1 + F2.
```

Output:

```
File  Edit  Settings  Run  Debug  Help
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- [fibonacci].
true.

?- fib(6, X).
X = 8 ;
false.

?- fib(5, 5).
true ;
false.

?- fib(5, 6).
false.

?-
```

Learning Outcomes

The Fibonacci program taught me about recursion. The main idea is that a problem is solved by calling a smaller version of itself.

- Base Cases are Super Important: I realized that without the starting points (fib(0,0) and fib(1,1)), the program would never stop. It's like a path without a beginning.
- Breaking Down a Problem: Instead of calculating the whole thing at once, I learned to break down the Fibonacci number into the two previous numbers. This makes complex problems much simpler.
- Thinking Logically: Prolog isn't like other languages where you give a computer a list of steps. Instead, I had to define what a Fibonacci number is, not how to calculate it step-by-step. The Prolog interpreter figures out the "how" for me.

Question

8.Write a PROLOG program to implement power (Num, Pow, Ans) : where Num is raised to the power Pow to get Ans.

Program:

power.pl

```
power(_, 0, 1).


power(Num, 1, Num).

power(Num, Pow, Ans) :-
    Pow > 1,
    NewPow is Pow - 1,
    power(Num, NewPow, TempAns),
    Ans is TempAns * Num.
```

Output:

```
SWI-Prolog (AMD64, Multi-threaded, version 8.0.3)
File  Edit  Settings  Run  Debug  Help
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.3)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- [power].
true.

?- power(2, 3, X).
X = 8 ;
false.

?- power(5, 4, Y).
Y = 625 ;
false.

?-
```

Learning Outcomes

The power program was a great way to practice the same ideas but with a different math problem.

- Recursion, Again!: This program reinforced the idea of recursion. To find Num to the power of Pow, I just needed to find Num to the power of Pow-1 and multiply it by Num. It's a simple rule that works for everything!
- The Power of Zero: The rule power(_, 0, 1) was a great reminder of a math rule I learned in school. It's a simple but vital piece of logic to make the program work correctly.
- Different Ways to Solve: I saw how both the Fibonacci and Power programs used the same fundamental recursive pattern. It showed me that recursion is a powerful tool that can be used to solve many different kinds of problems, from math to logic. I feel like I'm starting to see a pattern in how Prolog works, and that's really cool!

Question

9.PROLOG program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.

Program:

multi.pl

```
multi(_, 0, 0).
multi(N1, N2, R) :-
    N2 > 0,
    N_Dec is N2 - 1,
    multi(N1, N_Dec, R_Temp),
    R is N1 + R_Temp.
```

Output:

```
File  Edit  Settings  Run  Debug  Help
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).

?- [multi].
true.

?- multi(5, 3, R).
R = 15 ;
false.

?- multi(10, 0, R).
R = 0 ;
false.

?- multi(4, 4, 15).
false.

?- multi(1, 7, R).
R = 7
```

Learning Outcomes

- The rule multi(_, 0, 0). acts as the base case, serving as the stopping condition that prevents infinite recursion.
- Multiplication is defined recursively as repeated addition, where each recursive call adds the same number until the multiplier reaches zero.
- The is operator is used to make Prolog evaluate arithmetic expressions like N2 - 1 or N1 + R_Temp.
- In each recursive step, the problem simplifies as the multiplier (N2) decreases, bringing the computation closer to the base case.
- A temporary variable such as R_Temp stores intermediate results, allowing the final value to be computed step by step.

Question

10.Write a PROLOG program to implement memb(X, L): to check whether X is a member of L or not.

Program:

memb.pl

```
memb(X, [X|_]).
memb(X, [_|T]) :-
    memb(X, T).
```

Output:

```
File  Edit  Settings  Run  Debug  Help
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).

?- [memb].
true.

?- memb(b, [a, b, c, d]).
true ;
false.

?- memb(x, [a, b, c]).
false.

?- memb(X, [apple, banana, cherry]).
X = apple ;
X = banana ;
X = cherry ;
false.

?- memb(cat, []).
false.
```

Learning Outcomes

- Head and Tail Structure: Learned how Prolog divides a list into two parts — the first element called the Head and the remaining elements called the Tail — using the [Head|Tail] notation.
- Immediate Success Case: The rule memb(X, [X|_]). represents the success condition, returning true when the element X matches the first item of the list.
- Using the Underscore (_): The underscore symbol is used to ignore the current head of the list when it doesn't match, allowing the program to skip unnecessary elements.
- Recursion Through the Tail: The recursive call memb(X, T) continues searching through the Tail of the list until the desired element is found.
- End of the Search: There is no explicit rule for an empty list []; Prolog automatically infers failure when no elements remain to check.

Question

11. Write a PROLOG program to implement sumlist(L, S) so that S is the sum of a given list L.

Program:

program11.pl

```
sumlist([], 0).
sumlist([H|T], S) :- sumlist(T, ST), S is H + ST.
```

Output:

```
File  Edit  Settings  Run  Debug  Help
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).

?- [program11].
true.

?- sumlist([1,2,3,4], S).
S = 10.
```

Learning Outcomes

- I learned how recursion is used in PROLOG to process each element of a list.
- I understood how to use a base case ([]) to end recursion when the list becomes empty.
- I saw how arithmetic operations like S is H + ST combine results step-by-step.
- I now understand how list traversal works to compute the total sum of elements.

Question

12. Write a PROLOG program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.

Program:

program12.pl

```
evenlength([]).
evenlength([_,_|T]) :- evenlength(T).

oddlength([_]).
oddlength([_,_|T]) :- oddlength(T).
```

Output:

```
File  Edit  Settings  Run  Debug  Help
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).

?- [program12].
true.

?- evenlength([a,b,c,d]).
true.

?- oddlength([a,b,c]).
true
```

Learning Outcomes

- I learned how PROLOG uses pattern matching to classify lists based on their length.
- I understood how recursion alternates between even and odd by skipping two elements each time.
- I realized that a list with no elements is even-length, while a single-element list is odd-length.
- I now understand how base cases define when the recursion should stop.

Question

13. Write a PROLOG program to implement maxlist(L, M) so that M is the maximum number in the list.

Program:

program13.pl | program14.pl | program15.pl

```
maxlist([X],X).
maxlist([H|T],M):- maxlist(T,MT), M is max(H,MT).
```

Output:

```
?- [program13].
true.

?- maxlist([5,13,24,2],M).
M = 24 ;
```

Learning Outcomes

- I learned how to use recursion to go through every element in a list.
- I understood how to compare the head of a list with the maximum of the rest.
- I learned how the max/2 function helps to choose the larger number.
- I realized that the base case [X] stops recursion when only one element is left.
- I now know how to return the largest number from a list using Prolog logic.

Question

14. Write a PROLOG program to implement insert(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.

Program:

program13.pl | program14.pl | program15.pl

```
insert(I,1,L,[I|L]).
insert(I,N,[H|T],[H|R]):- N>1, N1 is N-1, insert(I,N1,T,R).
```

Output:

```
?- [program14].
true.

?- insert(x,3,[a,b,c,d],R).
R = [a, b, x, c, d] ;
```

Learning Outcomes

- I learned how to move through a list using recursion until I reach the desired position.
- I saw how pattern matching helps in breaking down the list into head and tail.
- I understood the use of arithmetic (N1 is N - 1) for counting positions.
- I learned how to rebuild a new list by inserting the new element in the right place.
- I realized that recursion and base cases together control where the insertion happens.

Question

15. Write a PROLOG program to implement delete(N, L, R) that removes the element on Nth position from a list L to generate a list R.

Program:

program13.pl | program14.pl | program15.pl

```
delete(1,[_|T],T).
delete(N,[H|T],[H|R]):- N>1, N1 is N-1, delete(N1,T,R).
```

Output:

```
?- [program15].
true.

?-  delete(2,[a,b,c,d],R).
R = [a, c, d] ;
```

Learning Outcomes

- I learned how to skip an element in a list using recursion.
- I understood how to use a base case to stop recursion when the desired position is reached.
- I saw how list decomposition ([H|T]) helps handle one element at a time.
- I learned how arithmetic operations (N1 is N - 1) help track positions during recursion.
- I now understand how to generate a new list without the deleted element.