Artificial Intelligence (01CE1702)
Lab Manual 24-25

Name: Dalsaniya Jay
ER No.: 92100103336
Calss: 7TC4

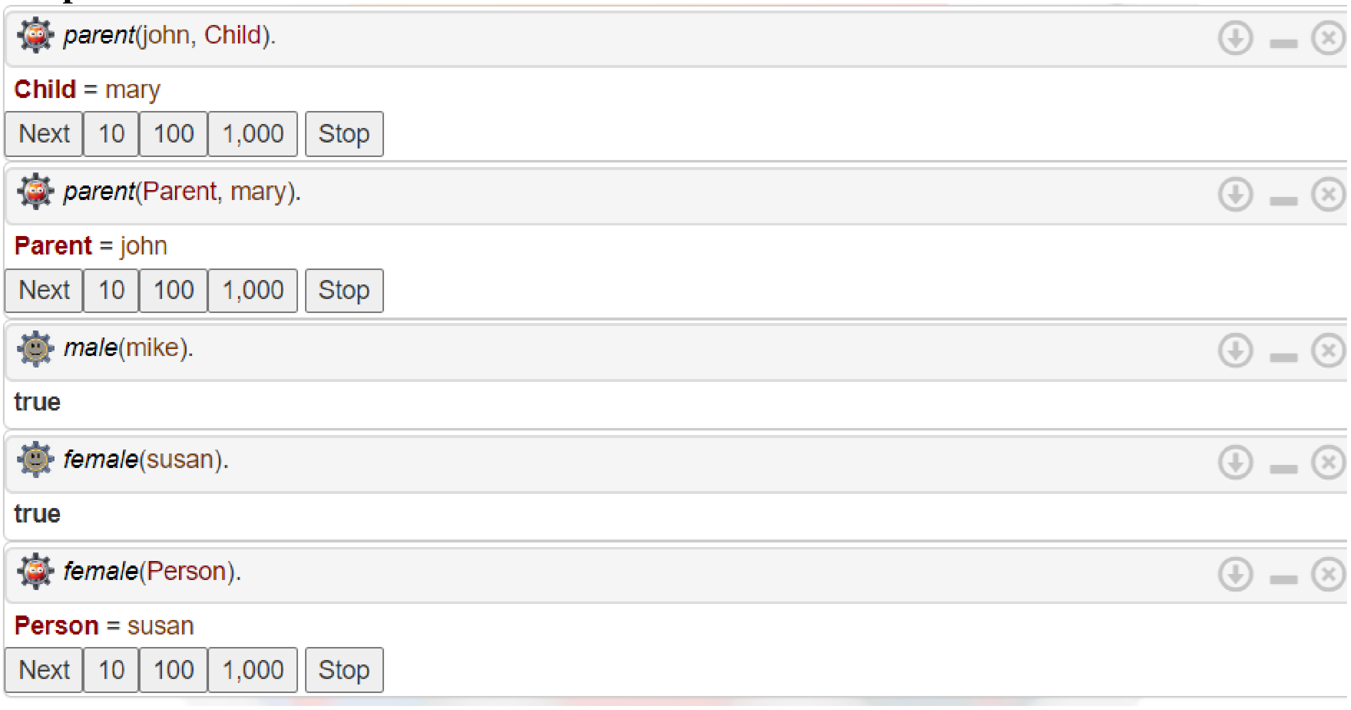
Lab	Program	Signature	Marks
1.	Write a prolog Program to understand the concept of facts and queries.		
2.	Write a prolog program to implement the following: a. Factorial of a given number b. Fibonacci of a given number		
3	Write a Prolog program to perform the following operations of the list, i) To display the element of the given list, ii) To check given element is in the list or not, iii) To print the last element of the list, Iv) To print the sum of the elements of the given list.		
4.	Implement a Family Tree and define the following predicates: 1)parent(X,Y) 2) Father(X,Y) 3) Mother(X,Y) 4) Sister(X,Y) 5)Brother(X,Y) 6)Grandfather(X,Y) 7)Grandmother(X,Y)		
5.	Assume given a set of facts of the form father(name1,name2) (name1 is the father of name2) Define a predicate cousin(X,Y) which holds iff X and Y are cousins. Define a predicate grandson(X,Y) which holds iff X is a grandson of Y. Define a predicate descendent(X,Y) which holds iff X is a descendent of Y. Define a predicate grandparent(X,Y) which holds iff X is a grandparent of Y. Consider the following genealogical tree: father(a,b). father(a,c). father(b,d). father(b,e). father(c,f). Say which answers, and in which order, are generated by your definitions for the following queries in Prolog: ?- cousin(X,Y). ?- grandson(X,Y). ?- descendent(X,Y). ?-grandparent(X,Y).		
6.	Write a program to solve Tower of Hanoi problem		
7.	Write a program to implement BFS for Water Jug problem/ 8 Puzzle problem or any AI search problem		
8.	Write a program to implement DFS for Water Jug problem/ 8 Puzzle problem or any AI search problem		
9.	Write a program to implement Single Player Game (Using Heuristic Function)		
10	Write a program to Implement A* Algorithm.		
11.	Implement the Mini Max algorithm for game playing		
12.	Write a program to solve N-Queens problem		
13	Develop an NLP application		
14	Implement Library for visual representations of text data		

Practical 1 : Write a prolog Program to understand the concept of facts and queries.

Program:

```
parent(john, mary).
parent(john, mike).
parent(susan, mary).
parent(susan, mike).
parent(mary, sophia).
parent(mary, james).
parent(paul, sophia).
parent(paul, james).

male(john).
male(mike).
male(paul).
male(james).

female(susan).
female(mary).
female(sophia).
```

Output :



```
parent(john, Child).
Child = mary

parent(Parent, mary).
Parent = john

male(mike).
true

female(susan).
true

female(Person).
Person = susan
```

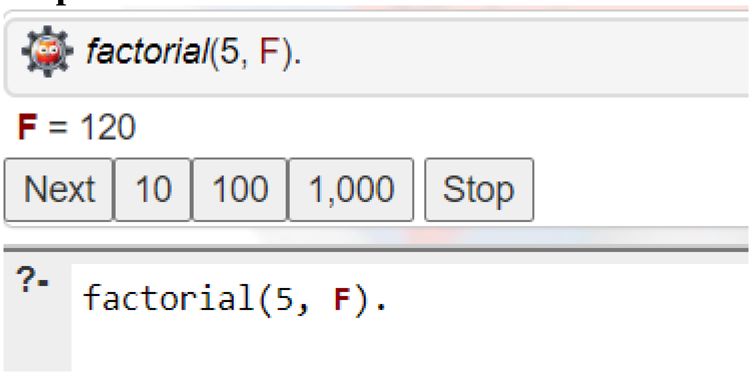
Practical 2 : Write a prolog program to implement the following:
a.Factorial of a given number
b.Fibonacci of a given number

program:

a) Factorial of a given number

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

output :



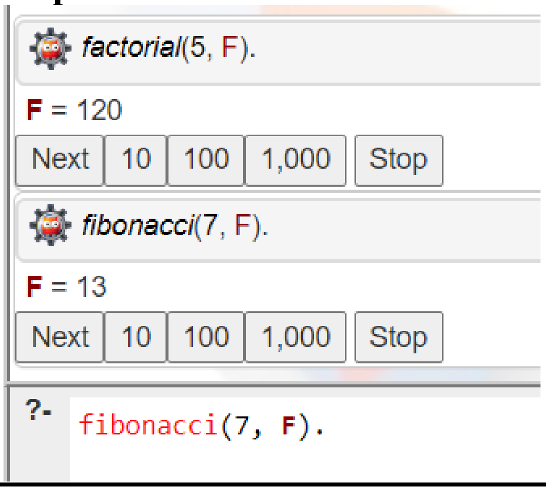
```
factorial(5, F).

F = 120
```

```
?- factorial(5, F).
```

b) Fibonacci of a given number

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

output :



```
factorial(5, F).

F = 120
```

```
fibonacci(7, F).

F = 13
```

```
?- fibonacci(7, F).
```

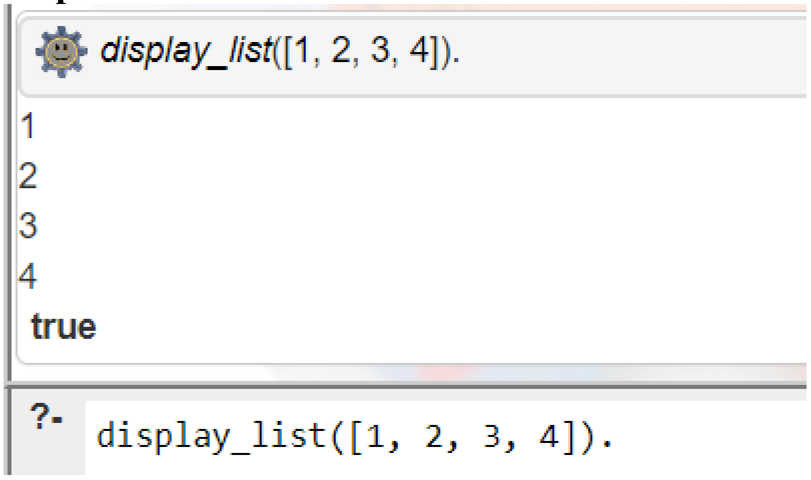
Practical 3 : Write a Prolog program to perform the following operations of the list,
i) To display the element of the given list,
ii) To check given element is in the list or not,
iii) To print the last element of the list,
Iv) To print the sum of the elements of the given list.

Program:

i) To display the element of the given list

```
display_list([]).
display_list([H|T]) :-
   write(H), nl,
   display_list(T).
```

output:

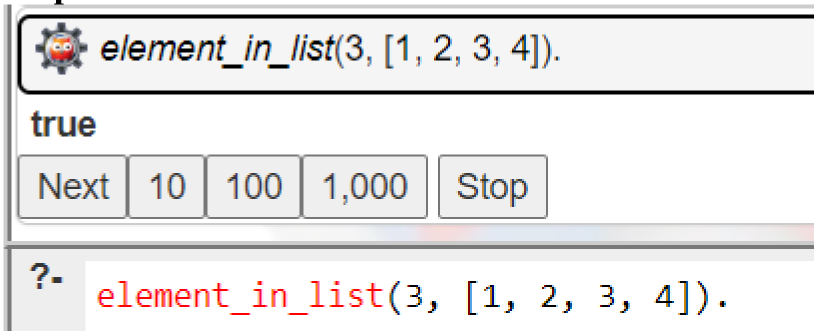


```
display_list([1, 2, 3, 4]).
1
2
3
4
true

?- display_list([1, 2, 3, 4]).
```

ii) To check given element is in the list or not

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

output:

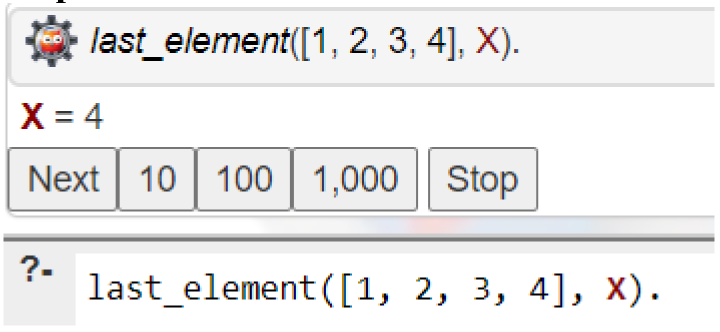


```
element_in_list(3, [1, 2, 3, 4]).
true

?- element_in_list(3, [1, 2, 3, 4]).
```

iii) To print the last element of the list

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

output:



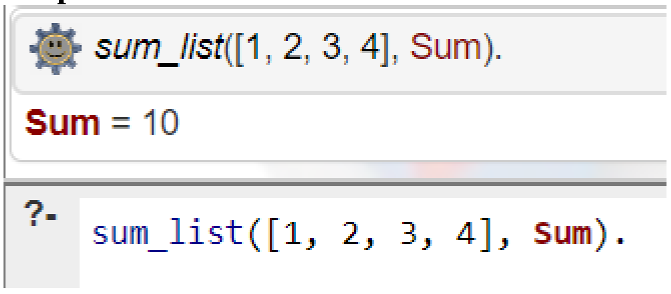
```
last_element([1, 2, 3, 4], X).
X = 4
```

```
?- last_element([1, 2, 3, 4], X).
```

iv) To print the sum of the elements of the given list.

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

Output:



```
sum_list([1, 2, 3, 4], Sum).
Sum = 10

?- sum_list([1, 2, 3, 4], Sum).
```

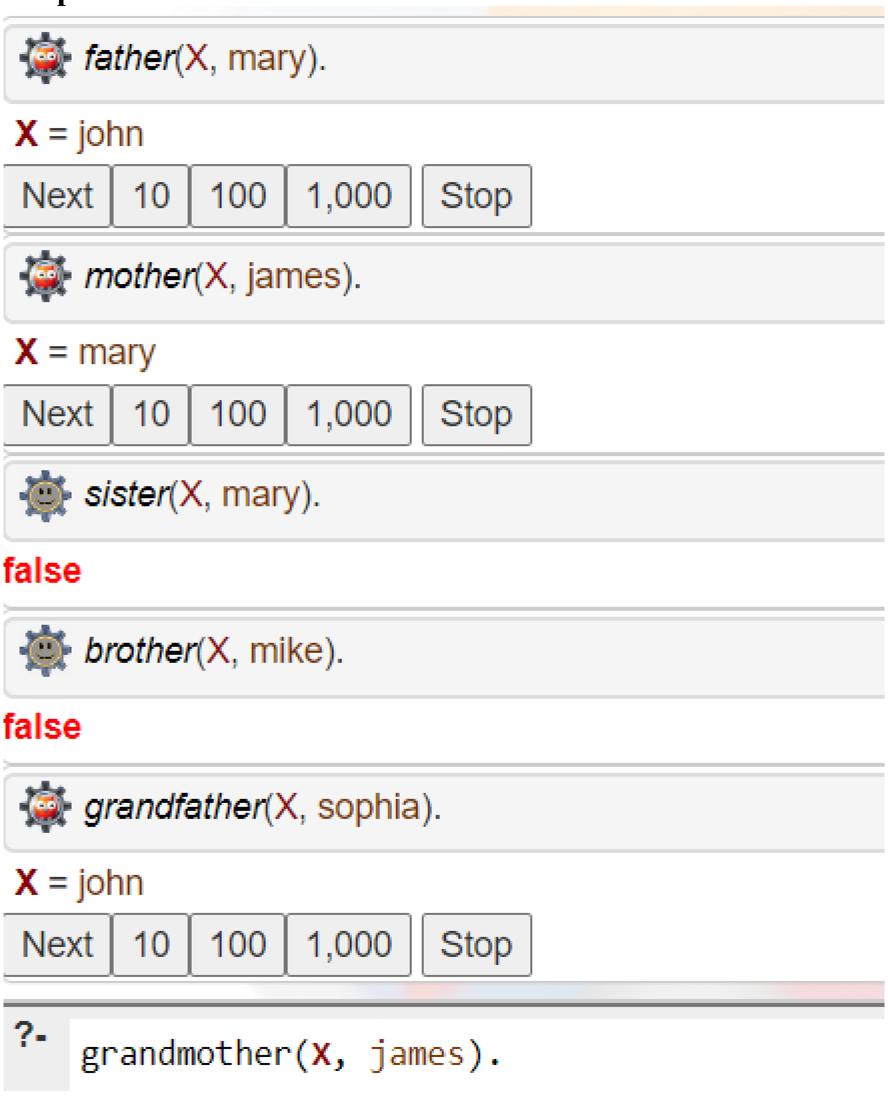
Practical 4: Implement a Family Tree and define the following predicates:

1)parent(X,Y)
2)Father(X,Y)
3)Mother(X,Y)
4)Sister(X,Y)
5)Brother(X,Y)
6)Grandfather(X,Y)
7)Grandmother(X,Y)

Program:

```
parent(john, mary).
parent(john, mike).

parent(susan, mary).
parent(susan, mike).

parent(mary, sophia).
parent(mary, james).

parent(paul, sophia).
parent(paul, james).

male(john).
male(mike).

male(paul).
male(james).

female(susan).
female(mary).

female(sophia).

father(X, Y) :- parent(X, Y), male(X).
mother(X, Y) :- parent(X, Y), female(X).

sister(X, Y) :- parent(Z, X), parent(Z, Y), female(X), X \= Y.
brother(X, Y) :- parent(Z, X), parent(Z, Y), male(X), X \= Y.

grandfather(X, Y) :- parent(X, Z), parent(Z, Y), male(X).
grandmother(X, Y) :- parent(X, Z), parent(Z, Y), female(X).
```

Output:



```
father(X, mary).
X = john
Next  10  100  1,000  Stop

mother(X, james).
X = mary
Next  10  100  1,000  Stop

sister(X, mary).
false

brother(X, mike).
false

grandfather(X, sophia).
X = john
Next  10  100  1,000  Stop

?- grandmother(X, james).
```

Practical 5: Assume given a set of facts of the form father(name1,name2) (name1 is the father of name2)
Define a predicate cousin(X,Y) which holds iff X and Y are cousins. Define a predicate grandson(X,Y) which holds iff X is a grandson of Y.
Define a predicate descendent(X,Y) which holds iff X is a descendent of Y. Define a predicate grandparent(X,Y) which holds iff X is a grandparent of Y.

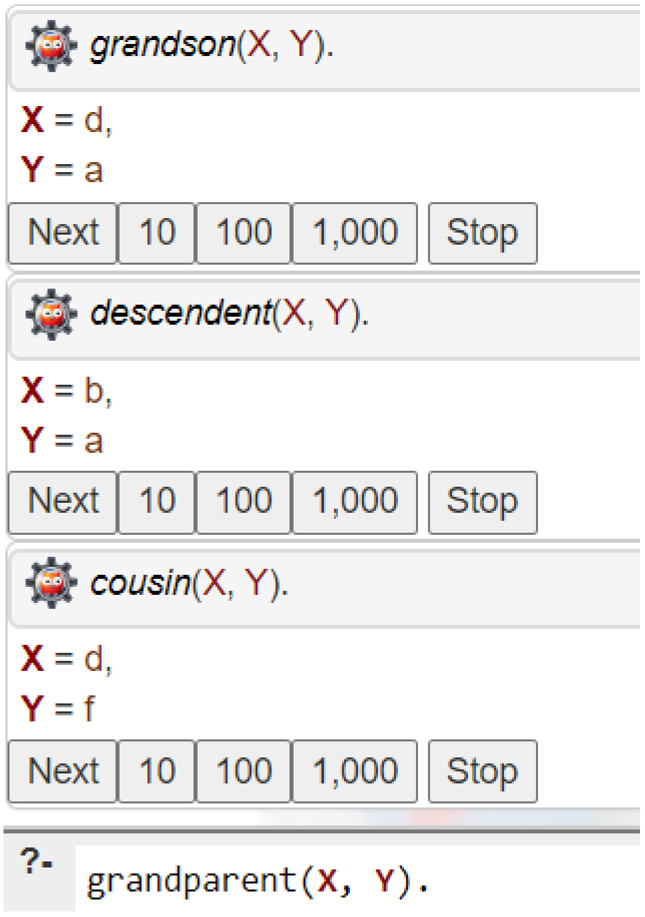
Consider the following genealogical tree:
father(a,b).
father(a,c).
father(b,d).
father(b,e).
father(c,f).
Say which answers, and in which order, are generated by your definitions for the following queries in Prolog:
?- cousin(X,Y).
?- grandson(X,Y).
?- descendent(X,Y).
?-grandparent(X,Y).

Program:

```
father(a, b).
father(a, c).

father(b, d).
father(b, e).

father(c, f).

cousin(X, Y) :-
  father(P1, X),
  father(P2, Y),

  father(GP, P1),
  father(GP, P2),
  P1 \= P2.

grandson(X, Y) :-
  father(Y, P),
  father(P, X).

descendent(X, Y) :-
  father(Y, X).
```

```
descendent(X, Y) :-
   father(Y, Z),
   descendent(X, Z).

grandparent(X, Y) :-
   father(X, P),
   father(P, Y).
```

Outout:



```
grandson(X, Y).
X = d,
Y = a
Next  10  100  1,000  Stop

descendent(X, Y).
X = b,
Y = a
Next  10  100  1,000  Stop

cousin(X, Y).
X = d,
Y = f
Next  10  100  1,000  Stop

?- grandparent(X, Y).
```