Artificial Intelligence (01CE1702)

Lab Manual 24-25

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

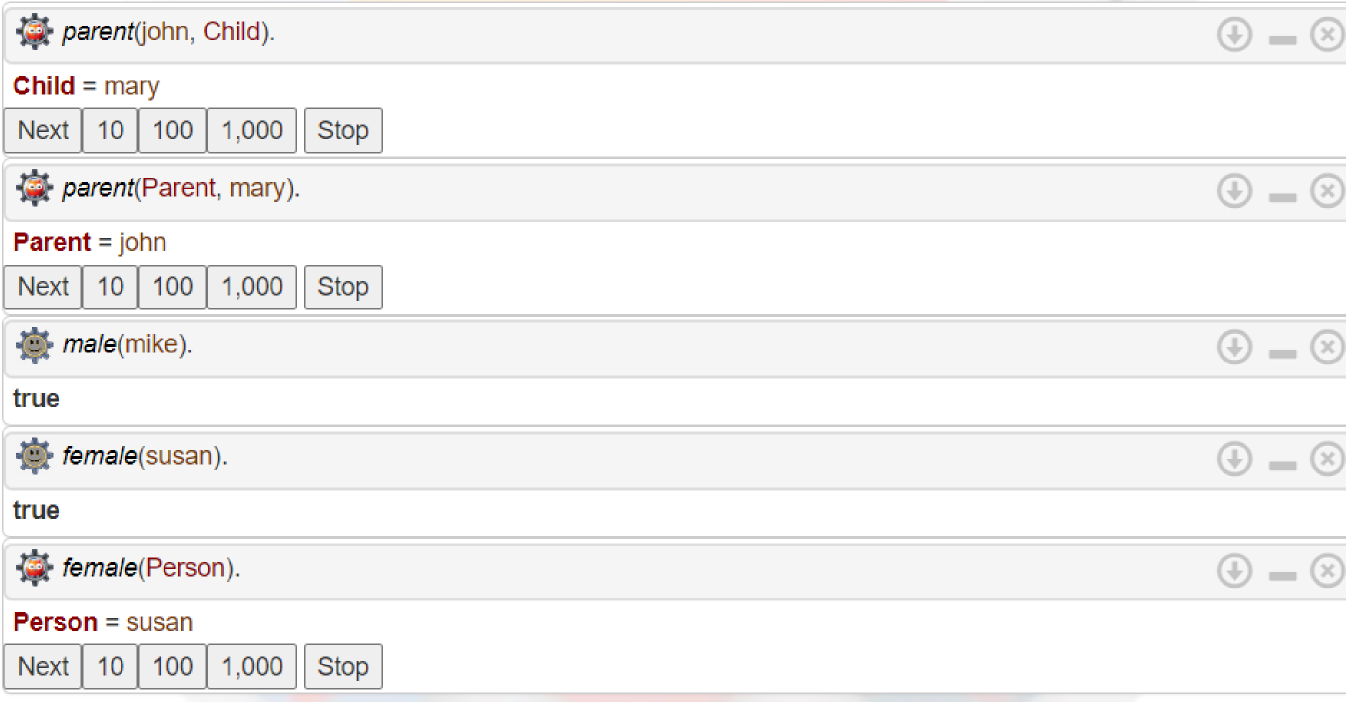
Lab	Program	Signature	Marks
1.	Write a prolog Program to understand the concept of facts and queries.		
2.	Write a prolog program to implement the following: a. Factorial of a given number b. Fibonacci of a given number		
3	Write a Prolog program to perform the following operations of the list, i) To 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
```

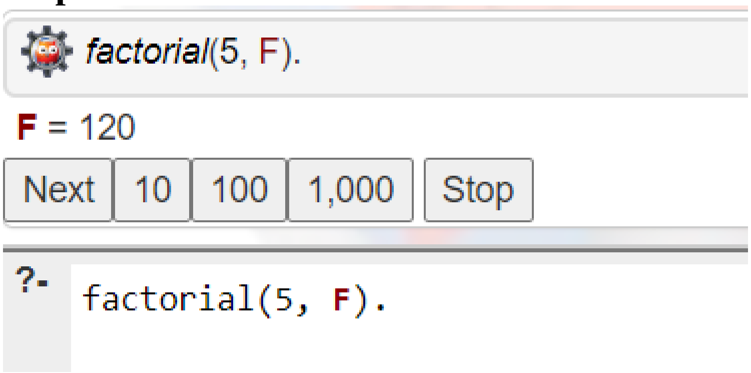
Practical 2 : Write a prolog program to implement the following:
a.Factorial of a given number
b.Fibonacci of a given number

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 :



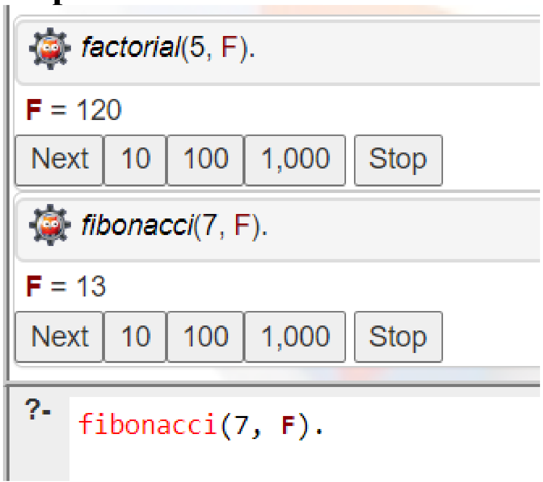
```
factorial(5, F).

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).
```

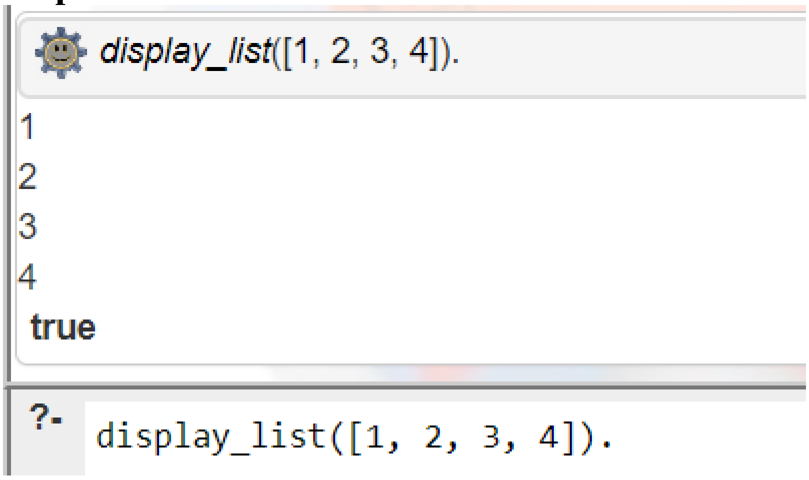
Practical 3 : Write a Prolog program to perform the following operations of the list,
i) To display the element of the given list,
ii) To check given element is in the list or not,
iii) To print the last element of the list,
Iv) To print the sum of the elements of the given list.

Program:

i) To display the element of the given list

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

output:

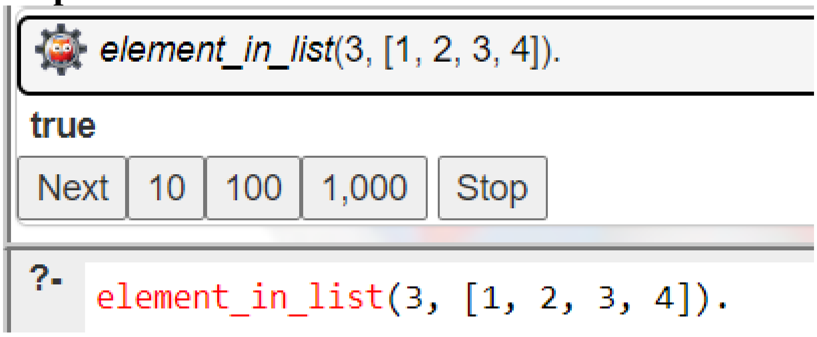


```
display_list([1, 2, 3, 4]).
1
2
3
4
true
?- display_list([1, 2, 3, 4]).
```

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:

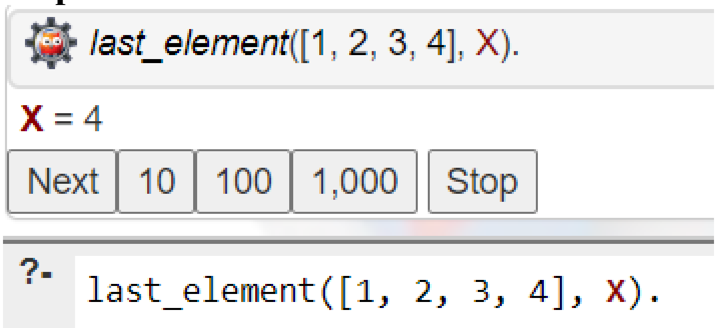


```
element_in_list(3, [1, 2, 3, 4]).
true
?- element_in_list(3, [1, 2, 3, 4]).
```

iii) To print the last element of the list

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

output:



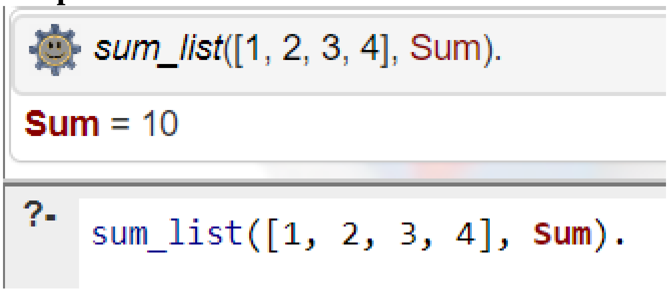
```
last_element([1, 2, 3, 4], X).

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).
```

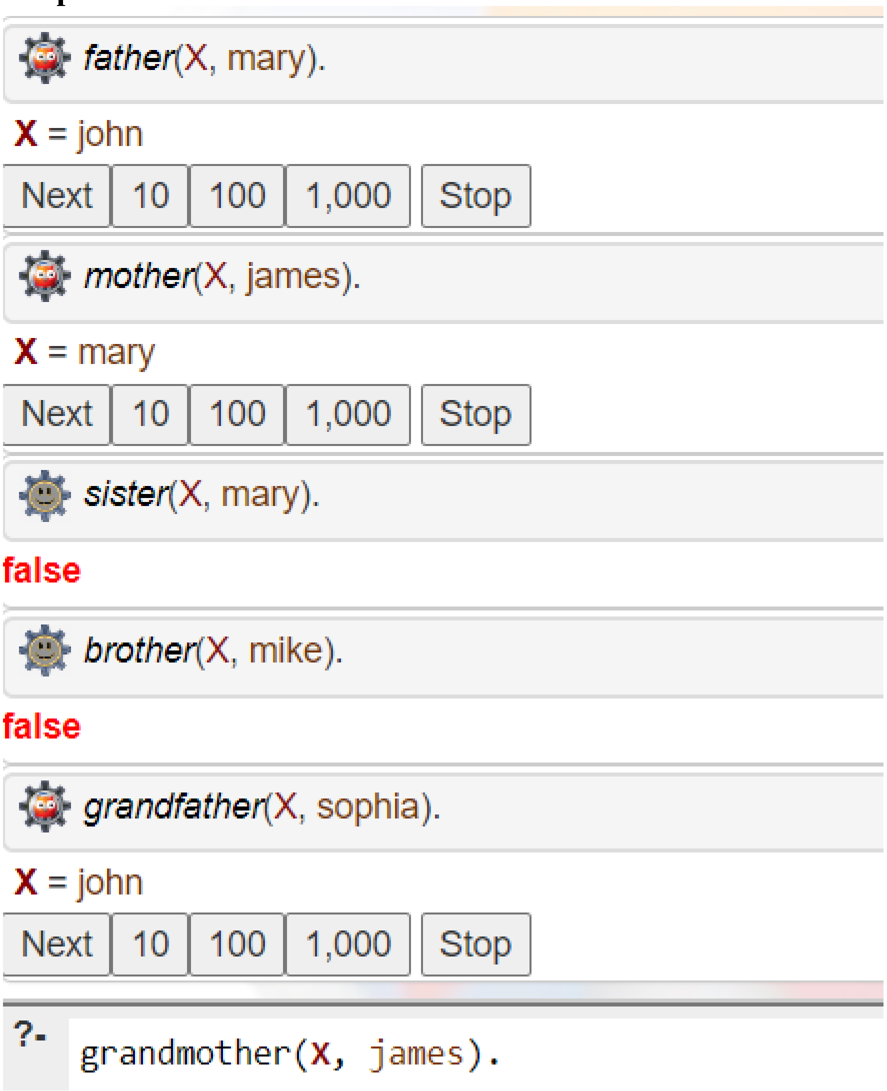
Practical 4: Implement a Family Tree and define the following predicates:

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

mother(X, james).
X = mary

sister(X, mary).
false

brother(X, mike).
false

grandfather(X, sophia).
X = john

?- 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:

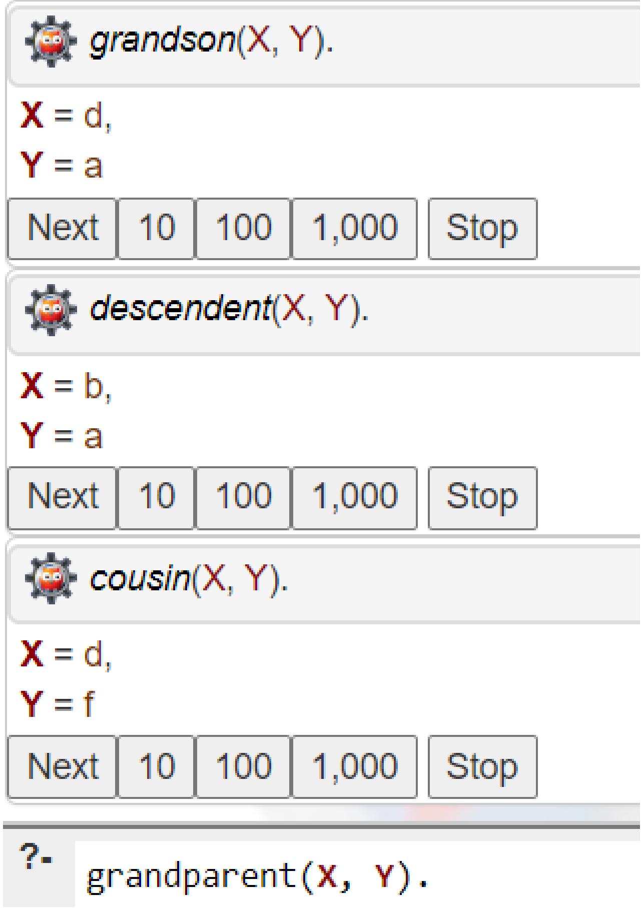
```
?- cousin(X,Y).
?- grandson(X,Y).
?- descendent(X,Y).
?-grandparent(X,Y).
```

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:



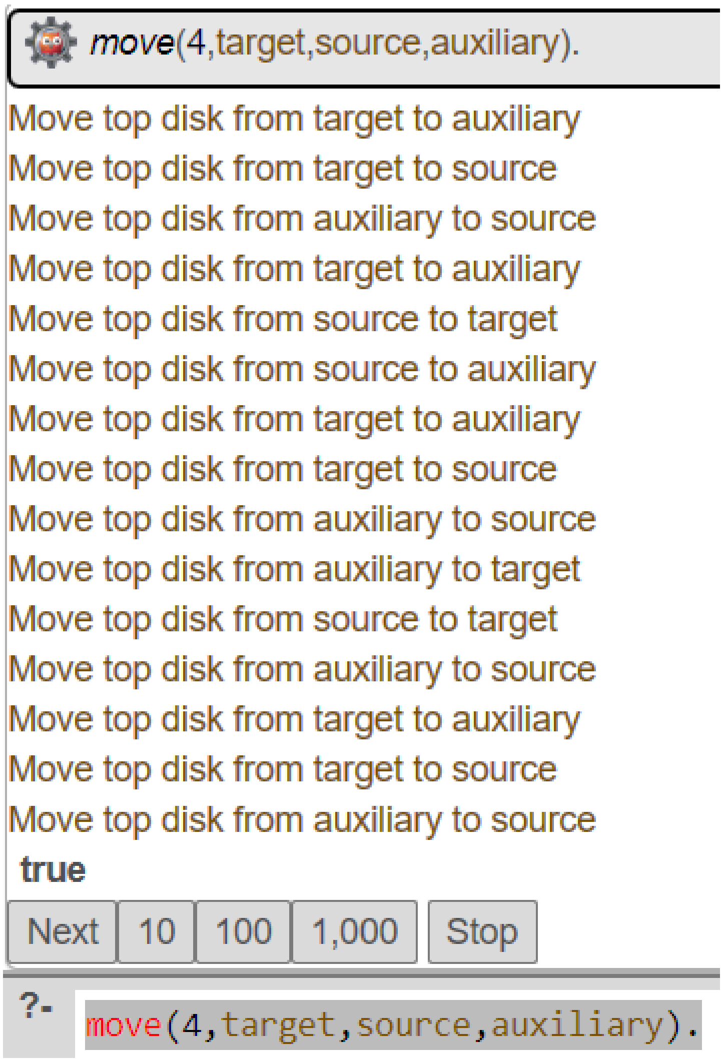
```
grandson(X, Y).
X = d,
Y = a

descendent(X, Y).
X = b,
Y = a

cousin(X, Y).
X = d,
Y = f

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

Practical 6 : Write a program to solve Tower of Hanoi problem

Program:

```
move(1, X, Y, _) :-
   write('Move top disk from '), write(X), write(' to '), write(Y), nl.

move(N, X, Y, Z) :-
   N > 1,
   M is N - 1,
   move(M, X, Z, Y),  % Move N-1 disks from Source to Auxiliary using Target as
auxiliary
   move(1, X, Y, _),  % Move the remaining disk from Source to Target
   move(M, Z, Y, X).  % Move the N-1 disks from Auxiliary to Target using Source as
auxiliary
```

Output:



```
move(4,target,source,auxiliary).
Move top disk from target to auxiliary
Move top disk from target to source
Move top disk from auxiliary to source
Move top disk from target to auxiliary
Move top disk from source to target
Move top disk from source to auxiliary
Move top disk from target to auxiliary
Move top disk from target to source
Move top disk from auxiliary to source
Move top disk from auxiliary to target
Move top disk from source to target
Move top disk from auxiliary to source
Move top disk from target to auxiliary
Move top disk from target to source
Move top disk from auxiliary to source
true
```

```
?- move(4,target,source,auxiliary).
```

Practical 7 : Water jug problem using BFS

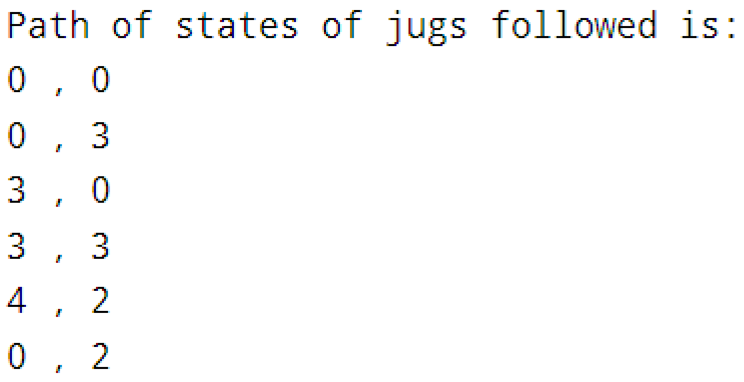
Program:

```
import java.util.*;

class Pair {
   int j1, j2;
   List<Pair> path;

   Pair(int j1, int j2) {
      this.j1 = j1;
      this.j2 = j2;
      path = new ArrayList<>();
   }

   Pair(int j1, int j2, List<Pair> _path) {
      this.j1 = j1;
      this.j2 = j2;
      path = new ArrayList<>(_path);
      path.add(new Pair(this.j1, this.j2));
   }
}

public class WaterJugProblem {
   public static void main(String[] args) throws java.lang.Exception {
      int jug1 = 4;
      int jug2 = 3;
      int target = 2;

      getPathIfPossible(jug1, jug2, target);
   }

   private static void getPathIfPossible(int jug1, int jug2, int target) {
      boolean[][] visited = new boolean[jug1 + 1][jug2 + 1];
      Queue<Pair> queue = new LinkedList<>();

      // Initial State: Both Jugs are empty so, initialise j1 j2 as 0 and put it in the path list
      Pair initialState = new Pair(0, 0);
      initialState.path.add(new Pair(0, 0));
      queue.offer(initialState);

      while (!queue.isEmpty()) {
         Pair curr = queue.poll();

         // Skip already visited states and overflowing water states
         if (curr.j1 > jug1 || curr.j2 > jug2 || visited[curr.j1][curr.j2]) {
            continue;
         }

         // Mark current jugs state as visited
```

```
visited[curr.j1][curr.j2] = true;

// Check if current state has already reached the target amount of water or not
if (curr.j1 == target || curr.j2 == target) {
    if (curr.j1 == target) {
        // If in our current state, jug1 holds the required amount of water, then we
        // empty the jug2 and push it into our path.
        curr.path.add(new Pair(curr.j1, 0));
    } else {
        // else, If in our current state, jug2 holds the required amount of water,
        // then we empty the jug1 and push it into our path.
        curr.path.add(new Pair(0, curr.j2));
    }

    int n = curr.path.size();
    System.out.println("Path of states of jugs followed is:");
    for (int i = 0; i < n; i++)
        System.out.println(curr.path.get(i).j1 + " , " + curr.path.get(i).j2);

    return;
}

// If we have not yet found the target, then we
// have three cases left:
// I. Fill the jug and Empty the other
// II. Fill the jug and let the other remain untouched
// III. Empty the jug and let the other remain untouched
// IV. Transfer amounts from one jug to another

// I. Fill the jug and Empty the other
queue.offer(new Pair(jug1, 0, curr.path));
queue.offer(new Pair(0, jug2, curr.path));

// II. Fill the jug and let the other remain untouched
queue.offer(new Pair(jug1, curr.j2, curr.path));
queue.offer(new Pair(curr.j1, jug2, curr.path));

// III. Empty the jug and let the other remain untouched
queue.offer(new Pair(0, curr.j2, curr.path));
queue.offer(new Pair(curr.j1, 0, curr.path));

// IV. Transfer water from one to another until one jug becomes empty or until
// one jug becomes full in this process

// Transferring water form jug1 to jug2
int emptyJug = jug2 - curr.j2;
int amountTransferred = Math.min(curr.j1, emptyJug);
int j2 = curr.j2 + amountTransferred;
int j1 = curr.j1 - amountTransferred;
queue.offer(new Pair(j1, j2, curr.path));
```

```
      // Transferring water form jug2 to jug1
      emptyJug = jug1 - curr.j1;
      amountTransferred = Math.min(curr.j2, emptyJug);
      j2 = curr.j2 - amountTransferred;
      j1 = curr.j1 + amountTransferred;
      queue.offer(new Pair(j1, j2, curr.path));
    }

    System.out.println("Not Possible to obtain target");
  }
}
```

Output:



```
Path of states of jugs followed is:
0 , 0
0 , 3
3 , 0
3 , 3
4 , 2
0 , 2
```

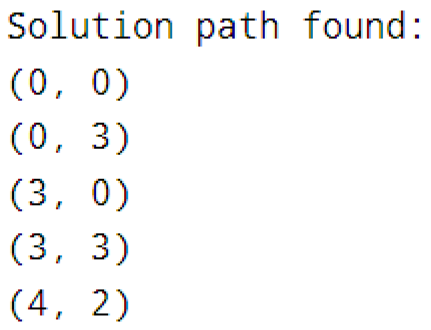
Practical 8 : Write a program to implement DFS for Water Jug problem/ 8 Puzzle problem or any AI search problem

Program:

```
def is_goal(state, target):
    return target in state

def get_successors(state, capacities):
    successors = []
    jug1, jug2 = state
    max1, max2 = capacities

    # Fill Jug1
    if jug1 < max1:
        successors.append((max1, jug2))
    # Fill Jug2
    if jug2 < max2:
        successors.append((jug1, max2))
    # Empty Jug1
    if jug1 > 0:
        successors.append((0, jug2))
    # Empty Jug2
    if jug2 > 0:
        successors.append((jug1, 0))
    # Pour Jug1 to Jug2
    if jug1 > 0 and jug2 < max2:
        pour_amount = min(jug1, max2 - jug2)
        successors.append((jug1 - pour_amount, jug2 + pour_amount))
    # Pour Jug2 to Jug1
    if jug2 > 0 and jug1 < max1:
        pour_amount = min(jug2, max1 - jug1)
        successors.append((jug1 + pour_amount, jug2 - pour_amount))

    return successors

def dfs_water_jug(start, capacities, target):
    stack = [start]
    visited = set()
    parent_map = {}

    while stack:
        state = stack.pop()

        if state in visited:
            continue

        visited.add(state)
```

```
            if is_goal(state, target):
                path = []
                while state:
                    path.append(state)
                    state = parent_map.get(state)
                return path[::-1]

            for successor in get_successors(state, capacities):
                if successor not in visited:
                    stack.append(successor)
                    parent_map[successor] = state

        return None

# Example usage
start_state = (0, 0)  # Both jugs are empty initially
jug_capacities = (4, 3)  # Capacity of jug1 is 4 liters, jug2 is 3 liters
target = 2  # The goal is to measure exactly 2 liters

solution_path = dfs_water_jug(start_state, jug_capacities, target)

if solution_path:
    print("Solution path found:")
    for state in solution_path:
        print(state)
else:
    print("No solution found.")
```

Output:



```
Solution path found:
(0, 0)
(0, 3)
(3, 0)
(3, 3)
(4, 2)
```

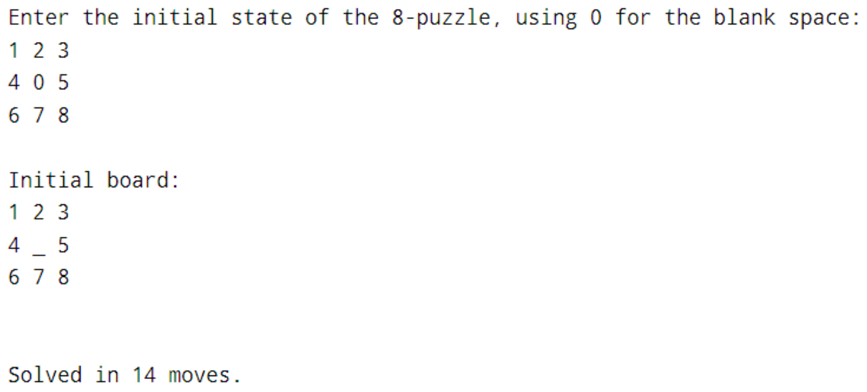
Practical 9 : Write a program to implement Single Player Game (Using Heuristic Function)

Program:

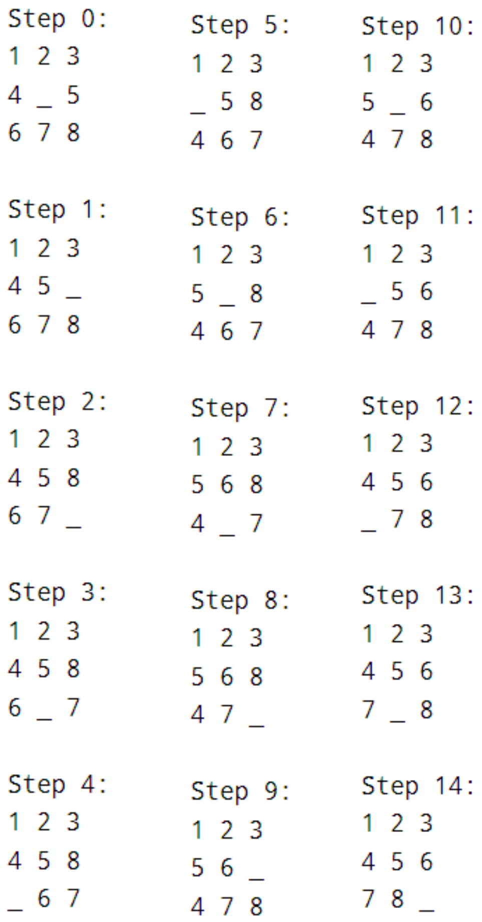
```
import heapq

class PuzzleState:
    def __init__(self, board, moves=0, previous=None):
        self.board = board
        self.moves = moves
        self.previous = previous
        self.blank_pos = self.find_blank()

    def find_blank(self):
        for i in range(3):
            for j in range(3):
                if self.board[i][j] == 0:
                    return (i, j)

    def __lt__(self, other):
        return self.priority() < other.priority()

    def priority(self):
        return self.moves + self.manhattan_distance()

    def manhattan_distance(self):
        distance = 0
        for i in range(3):
            for j in range(3):
                if self.board[i][j] != 0:
                    x, y = divmod(self.board[i][j] - 1, 3)
                    distance += abs(x - i) + abs(y - j)
        return distance

    def is_goal(self):
        goal = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
        return self.board == goal

    def generate_successors(self):
        successors = []
        x, y = self.blank_pos
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
        for dx, dy in directions:
            new_x, new_y = x + dx, y + dy
            if 0 <= new_x < 3 and 0 <= new_y < 3:
                new_board = [row[:] for row in self.board]
                new_board[x][y], new_board[new_x][new_y] = new_board[new_x][new_y],
new_board[x][y]
```

```
                successors.append(PuzzleState(new_board, self.moves + 1, self))
        return successors

def print_board(board):
    for row in board:
        print(" ".join(str(num) if num != 0 else "_" for num in row))
    print()

def a_star_search(initial_board):
    start_state = PuzzleState(initial_board)
    open_set = []

    heapq.heappush(open_set, start_state)
    closed_set = set()

    while open_set:
        current_state = heapq.heappop(open_set)

        if current_state.is_goal():
            return current_state

        closed_set.add(tuple(map(tuple, current_state.board)))

        for successor in current_state.generate_successors():
            if tuple(map(tuple, successor.board)) not in closed_set:
                heapq.heappush(open_set, successor)

    return None

def reconstruct_path(state):
    path = []
    while state:
        path.append(state.board)
        state = state.previous
    return path[::-1]

def main():
    print("Enter the initial state of the 8-puzzle, using 0 for the blank space:")
    initial_board = []
    for _ in range(3):
        row = list(map(int, input().split()))
        initial_board.append(row)

    print("\nInitial board:")
    print_board(initial_board)

    solution = a_star_search(initial_board)
    if solution:
        path = reconstruct_path(solution)
        print(f"\nSolved in {len(path) - 1} moves.\n")
        for i, step in enumerate(path):
            print(f"Step {i}:")
            print_board(step)
    else:
        print("No solution found.")

if __name__ == "__main__":
    main()
```

Output :



```
Enter the initial state of the 8-puzzle, using 0 for the blank space:
1 2 3
4 0 5
6 7 8

Initial board:
1 2 3
4 _ 5
6 7 8


Solved in 14 moves.
```



```
Step 0:     Step 5:     Step 10:
1 2 3       1 2 3       1 2 3
4 _ 5       _ 5 8       5 _ 6
6 7 8       4 6 7       4 7 8

Step 1:     Step 6:     Step 11:
1 2 3       1 2 3       1 2 3
4 5 _       5 _ 8       _ 5 6
6 7 8       4 6 7       4 7 8

Step 2:     Step 7:     Step 12:
1 2 3       1 2 3       1 2 3
4 5 8       5 6 8       4 5 6
6 7 _       4 _ 7       _ 7 8

Step 3:     Step 8:     Step 13:
1 2 3       1 2 3       1 2 3
4 5 8       5 6 8       4 5 6
6 _ 7       4 7 _       7 _ 8

Step 4:     Step 9:     Step 14:
1 2 3       1 2 3       1 2 3
4 5 8       5 6 _       4 5 6
_ 6 7       4 7 8       7 8 _
```