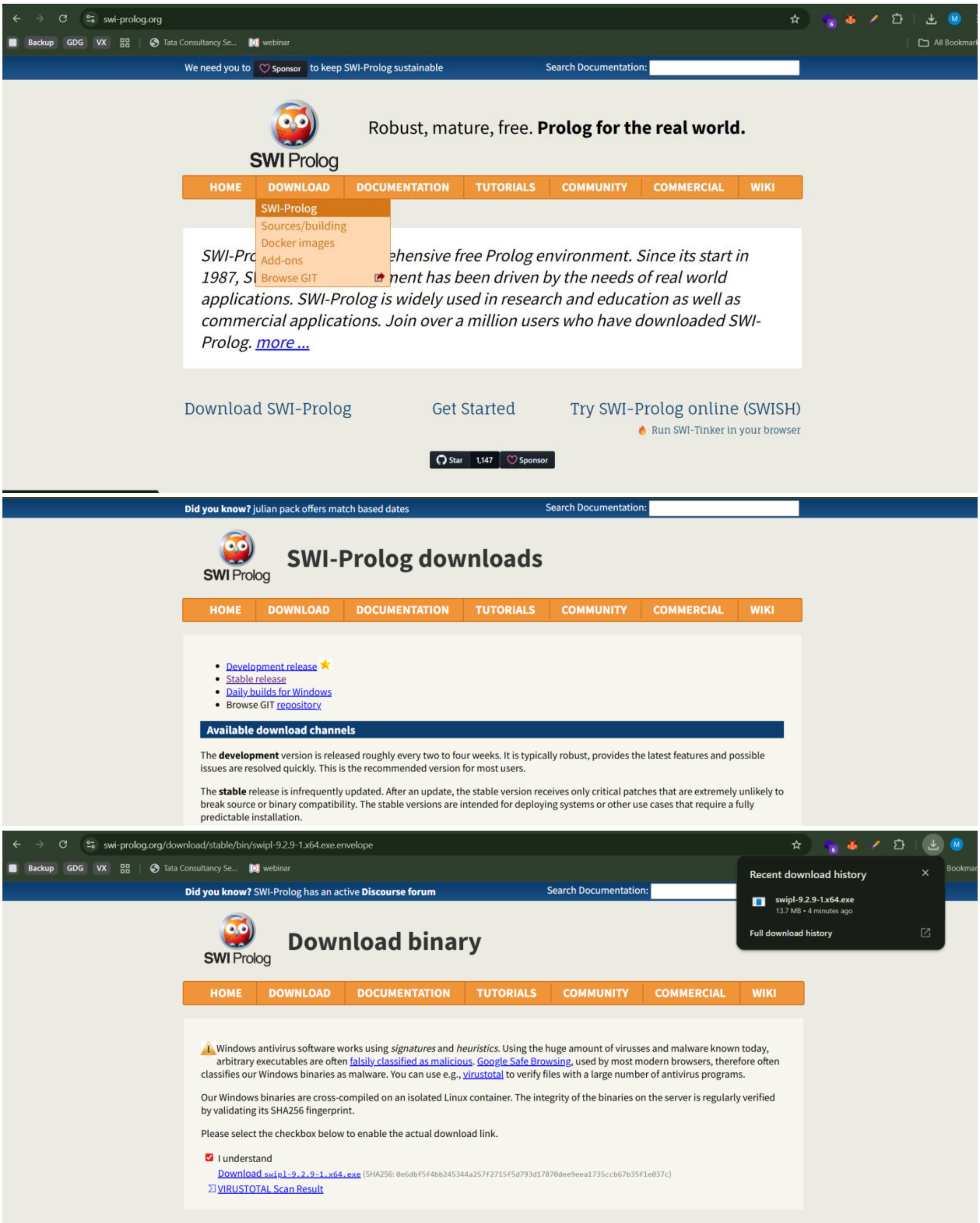
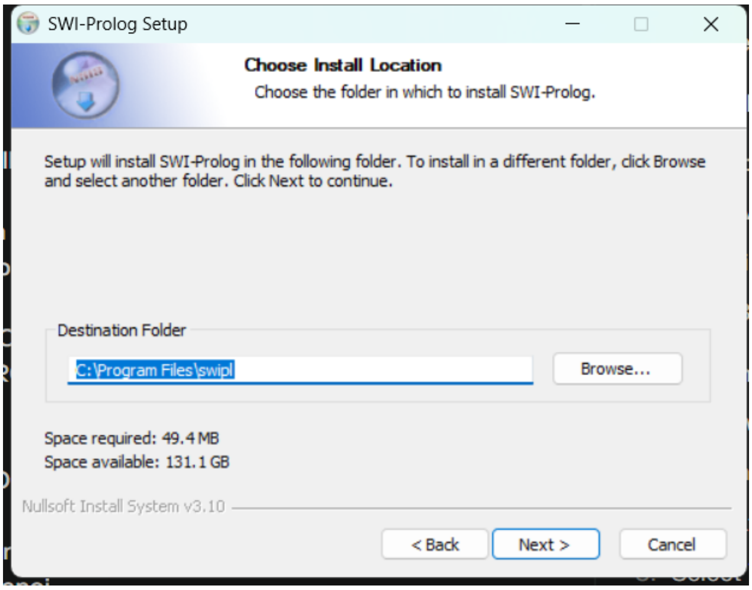
Practical 01 =>
Prolog installation.

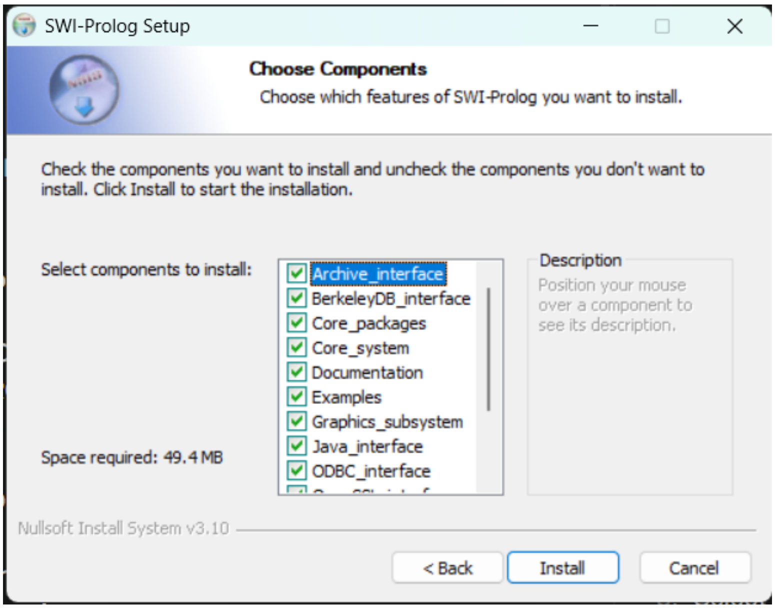
Step1) Visit the official website: www.swi-prolog.org and download the installer.



Name: Mayank Mandar Chandratre
Roll No. 07

Step 2) Run the setup and complete the installation.





Step 3) Verify Installation.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\DELL> swipl
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).

1 ?- halt
.
PS C:\Users\DELL> |
```

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 02 =>

Basic Programs in Prolog.

1)

program1.pl

```prolog
cat(tom).
loves_to_eat(kunal,pasta).
of_color(hair,black).
loves_to_play_games(nawaz).
lazy(pratyusha).
like(ram,mango).
girl(seema).
red(rose).
likes(bill,gold).
owns(john,gold).

```

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).

?-
% d:/prolog_programs/program1.pl compiled 0.00 sec, 10 clauses
?-
|    cat(Who).
Who = tom.

?- lazy(Who).
Who = pratyusha.

?- loves_to_play_games(Who).
Who = nawaz.

?- loves_to_eat(kunal,What).
What = pasta.

?-
```

Name: Mayank Mandar Chandratre

Roll No. 07

2)

program2.pl

```prolog
symptom(chicken_pox,high_fever).
symptom(chicken_pox,chills).
symptom(flu,chills).
symptom(cold,mild_body_ache).
symptom(flu,severe_body_ache).
symptom(cold,runny_nose).
symptom(flu,runny_nose).
symptom(flu,moderate_cough).
girl(priya).
girl(tiyasha).
girl(jaya).
can_cook(priya).
can_cook(jaya).
can_cook(tiyasha).
likes(priya, jaya) :- can_cook(jaya).
likes(priya, tiyasha) :- can_cook(tiyasha).
```

```
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).

?-
% d:/prolog_programs/program2.pl compiled 0.00 sec, 16 clauses
?-
|    symptom(flu,What).
What = chills .

?- symptom(flu,What).
What = chills ;
What = severe_body_ache ;
What = runny_nose ;
What = moderate_cough.

?- can_cook(X).
X = priya ;
X = jaya ;
X = tiyasha.

?- likes(priya,X).
X = jaya ;
X = tiyasha.

?-
```

Name: Mayank Mandar Chandratre
Roll No. 07

3)

program3.pl

```prolog
sing_a_song(ananya).
listens_to_music(rohit).
listens_to_music(ananya):-sing_a_song(ananya).
happy(ananya):-sing_a_song(ananya).
happy(rohit):-listens_to_music(rohit).
playes_guitar(rohit):-listens_to_music(rohit).
dog(rottweiler).
cat(sphynx).
dog(poodle).
dog(bulldog).
cat(bengali).
dog(dobernmann).
cat(himalayan).
cat(singapura).

```

```
Warning:     Use :- discontiguous cat/1. to suppress this message
% d:/prolog_programs/program3.pl compiled 0.00 sec, 14 clauses
?-
|     happy(rohit).
true.

?- sing_a_song(rohit).
false.

?- sing_a_song(ananya).
true.

?- dog(A).
A = rottweiler
Unknown action: l (h for help)
Action? .

?-
|     █
```

4)

program4.pl

```prolog
in(room,banana).
at(ceiling,banana).
strong(monkey).
grasp(monkey).
climb(monkey,chair).
push(monkey,chair):-
strong(monkey).
under(banana,chair):-
push(monkey,chair).
canreach(banana,monkey):-
at(floor,banana);
at(ceiling,banana),
under(banana,chair),
climb(monkey,chair).
canget(banana,monkey):-
canreach(banana,monkey),grasp(monkey).
```

```
% d:/prolog_programs/program4.pl compiled 0.00 sec, 9 clauses
|
|     cagnet(banana,monkey).
Correct to: "canget(banana,monkey)"? yes
true.

?- canreach(banana,monkey).
true.

?- strong(monkey).
true.

?- push(banana,monkey).
false.

?-
```

Name: Mayank Mandar Chandratre
Roll No. 07

5)

program5.pl

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

```
% d:/prolog_programs/program5.pl compiled 0.00 sec, 2 clauses
?-
|    factorial(10,X).
X = 3628800.

?- factorial(7,X).
X = 5040.

?-
```

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 03 =>

Arithmetic Programs in Prolog.

1)

arith1.pl

```
% Check if a number is prime
is_prime(2) :- !.
is_prime(N) :-
    N > 2,
    N mod 2 =\= 0,
    \+ has_odd_divisor(N, 3).

has_odd_divisor(N, D) :-
    D * D =< N,
    (N mod D =:= 0 ; D1 is D + 2, has_odd_divisor(N, D1)).

% Find all primes up to N
primes_up_to(N, Primes) :-
    findall(X, (between(2, N, X), is_prime(X)), Primes).
```

```
% d:/prolog_programs/arith1.pl compiled 0.00 sec, 4 clauses
?-
|    is_prime(3).
true.

?- is_prime(10).
false.

?-
```

Name: Mayank Mandar Chandratre

Roll No. 07

2)

arith2.pl

```
% Calculate GCD using Euclidean algorithm
gcd(A, 0, A) :- !.
gcd(A, B, G) :-
    B > 0,
    R is A mod B,
    gcd(B, R, G).

% Calculate LCM
lcm(A, B, L) :-
    gcd(A, B, G),
    L is (A * B) // G.
```

```
?- gcd(176, 178, A).
A = 2.

?- gcd(39, 13, A).
A = 13.

?- lcm(39, 13, A).
A = 39.

?- lcm(15, 13, A).
A = 195.

?- 
```

Name: Mayank Mandar Chandratre
Roll No. 07

3)

arith3.pl

```prolog
% Program: Count to 10

% Base case: When the number is 10, print it and stop.
count_to_10(10) :-
    write(10), nl.

% Recursive step: Print X, calculate Y = X + 1, then count(Y).
count_to_10(X) :-
    write(X), nl,
    Y is X + 1,
    count_to_10(Y).
```

```
% d:/prolog_programs/arith3.pl compiled 0.00 sec, 2 clauses
?- count_to_10(7).
7
8
9
10
true
Unknown action: □ (h for help)
Action? .

?-
|    count_to_10(1).
1
2
3
4
5
6
7
8
9
10
true .

?-
|
```

Name: Mayank Mandar Chandratre
Roll No. 07

4)

arith4.pl

```prolog
% Calculator predicate
calc(Operator, X, Y, Result) :-
    Operator = '+', Result is X + Y;
    Operator = '-', Result is X - Y;
    Operator = '*', Result is X * Y;
    Operator = '/', Y \= 0, Result is X / Y;
    Operator = 'mod', Result is X mod Y.
```

```
?- calc('-',5,4,R).
R = 1 .

?- calc('/',5,4,R).
R = 1.25 .

?- calc('+',5,4,R).
R = 9 .

?- calc('*',5,4,R).
R = 20 .

?-
```

Name: Mayank Mandar Chandratre
Roll No. 07

5)

arith5.pl

```prolog
% Fibonacci 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.

% Print first N fibonacci numbers
print_fibonacci(0) :- !.
print_fibonacci(N) :-
    N > 0,
    N1 is N - 1,
    print_fibonacci(N1),
    fibonacci(N1, F),
    write(F), write(' '), nl.
```

```
% d:/prolog_programs/arith5.pl compiled 0.00 sec, 5 clauses
?- print_fibonacci(5).
0
1
1
2
3
true.

?- print_fibonacci(10).
0
1
1
2
3
5
8
13
21
34
true.

?-
```

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 04 =>

a. Program to generate Family tree using PROLOG

family_tree.pl

```prolog
parent(mandar, mayank).
parent(mandar, mihir).
parent(malvika, mayank).
parent(malvika, mihir).

% --- Rules (Logic) from Assignment 4 ---

% X is a grandparent of Y if X is a parent of Z, and Z is a parent of Y
grandparent(X, Y) :-
    parent(X, Z),
    parent(Z, Y).

% X and Y are siblings if they share a parent Z and X is not Y
sibling(X, Y) :-
    parent(Z, X),
    parent(Z, Y),
    X \= Y.

% X is the father of Y if X is a parent of Y and X is male
father(X, Y) :-
    parent(X, Y),
    male(X).

% X is the mother of Y if X is a parent of Y and X is female
mother(X, Y) :-
    parent(X, Y),
    female(X).

% X is an ancestor of Y (Recursive definition)
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

% X is a descendant of Y
descendant(X, Y) :- ancestor(Y, X).
```

```
?- grandparent(X,mayank).
X = madan ;
X = manjiri ;
false.

?- grandparent(madan,mandar).
false.

?- ancestor(X,mihir).
X = mandar ;
X = malvika ;
X = madan ;
X = manjiri .

?- descendent(madan,X).
Correct to: "descendant(madan,X)"? yes
false.

?- descendent(X, manjiri).
Correct to: "descendant(X,manjiri)"? yes
X = mandar ;
X = mayank ;
X = mihir ;
false.

?-
```

Conclusion: Hence we execute a program in prolog for family tree.

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 05 =>

a. Write a program in prolog to solve Monkey banana problem

pr5.pl

```prolog
initial_state(state(at_door, at_door, onfloor, hanging)).
goal_state(state(_, _, _, grasped)).

% --- Move Rules ---

% 1. Walk from Door to Banana
move(state(at_door, CrateLoc, onfloor, Banana),
     state(at_banana, CrateLoc, onfloor, Banana), 'Walk to Banana').

% 2. Walk from Banana to Door
move(state(at_banana, CrateLoc, onfloor, Banana),
     state(at_door, CrateLoc, onfloor, Banana), 'Walk to Door').

% 3. Push Crate from Door to Banana (FIXED: Monkey moves WITH crate)
move(state(at_door, at_door, onfloor, Banana),
     state(at_banana, at_banana, onfloor, Banana), 'Push Crate to Banana').

% 4. Push Crate from Banana to Door (FIXED: Monkey moves WITH crate)
move(state(at_banana, at_banana, onfloor, Banana),
     state(at_door, at_door, onfloor, Banana), 'Push Crate to Door').

% 5. Climb Crate
move(state(at_banana, at_banana, onfloor, Banana),
     state(at_banana, at_banana, oncrate, Banana), 'Climb Crate').

% 6. Grasp Banana
move(state(at_banana, at_banana, oncrate, hanging),
     state(at_banana, at_banana, oncrate, grasped), 'Grasp Banana').

% --- Solver (FIXED with Cycle Detection) ---

% Entry point for the user
solve_problem :-
    initial_state(Start),
    solve(Start, [], Path),   % Start with empty visited list
    write('Solution Path:'), nl,
    print_path(Path).

% Base Case: Goal reached
solve(State, _, []) :-
    goal_state(State).

% Recursive Step: Try a move
solve(CurrentState, Visited, [Action | Path]) :-
    move(CurrentState, NextState, Action),
    \+ member(NextState, Visited),
    solve(NextState, [NextState | Visited], Path).

% Helper to print the path neatly
print_path([]).
print_path([Action | Rest]) :-
    write(' -> '), write(Action), nl,
    print_path(Rest).
```

Name: Mayank Mandar Chandratre
Roll No. 07

```
% d:/prolog_programs/pr5.pl compiled 0.00 sec, 3 clauses
?- solve_problem.
Solution Path:
 -> Walk to Banana
 -> Walk to Door
 -> Push Crate to Banana
 -> Climb Crate
 -> Grasp Banana
true
```

b. Write a program in prolog to solve Tower of Hanoi

hanoi.pl

```
hanoi(1, Source, Target, _):-
    write('Move disk 1 from '), write(Source), write(' to '), write(Target), nl.

hanoi(N, Source, Target, Auxiliary):-
    N > 1,
    M is N - 1,
    hanoi(M, Source, Auxiliary, Target),
    write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Target), nl,
    hanoi(M, Auxiliary, Target, Source).
```

hanoi.pl

```
hanoi(1, Source, Target, _):-
    write('Move disk 1 from '), write(Source), write(' to '), write(Target), nl.

hanoi(N, Source, Target, Auxiliary):-
    N > 1,
    M is N - 1,
    hanoi(M, Source, Auxiliary, Target),
    write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Target), nl,
    hanoi(M, Auxiliary, Target, Source).
```


Conclusion: Hence we solved Monkey Banana Problem and Tower Of Hanoi in Prolog.

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 06 =>

a. Write a program in prolog to solve 8 Puzzle problems

puzzle.pl

```prolog
% ==========================================
% 8-Puzzle with Depth Limit (Prevents Loops)
% ==========================================

goal([1,2,3,
      4,5,6,
      7,8,0]).

% --- List Helpers ---
set_elem([_|T], 0, X, [X|T]).
set_elem([H|T], I, X, [H|R]) :-
    I > 0, I1 is I - 1, set_elem(T, I1, X, R).

swap(List, I, J, Result) :-
    nth0(I, List, A), nth0(J, List, B),
    set_elem(List, I, B, Temp),
    set_elem(Temp, J, A, Result).

% --- Move Logic ---
move(State, NextState) :-
    nth0(Blank, State, 0),
    move_blank(Blank, NewBlank),
    swap(State, Blank, NewBlank, NextState).

move_blank(0,1). move_blank(0,3).
move_blank(1,0). move_blank(1,2). move_blank(1,4).
move_blank(2,1). move_blank(2,5).
move_blank(3,0). move_blank(3,4). move_blank(3,6).
move_blank(4,1). move_blank(4,3). move_blank(4,5). move_blank(4,
move_blank(5,2). move_blank(5,4). move_blank(5,8).
move_blank(6,3). move_blank(6,7).
move_blank(7,4). move_blank(7,6). move_blank(7,8).
move_blank(8,5). move_blank(8,7).

% --- SOLVER (Iterative Deepening) ---

solve(Start) :-
    % Try to find path of length 1, then 2, then 3... up to 15
    between(1, 15, Limit),
    write('Searching with depth limit: '), write(Limit), nl,
    path(Start, [Start], Limit).

path(State, Visited, _) :-
    goal(State),
    write('Goal State Reached!'), nl,
```

puzzle.pl


```prolog
path(State, Visited, _) :-
    goal(State),
    write('Goal State Reached!'), nl,
```


```prolog
    print_path(Visited).

path(State, Visited, Limit) :-
    Limit > 0,
    move(State, Next),
    \+ member(Next, Visited),
    NewLimit is Limit - 1,
    path(Next, [Next | Visited], NewLimit).

% --- Print Path in Correct Order ---
print_path(Visited) :-
    reverse(Visited, Path),
    print_steps(Path).

print_steps([]).
print_steps([H|T]) :-
    print_state(H),
    print_steps(T).

print_state([A,B,C,D,E,F,G,H,I]) :-
    write(A), write(' '), write(B), write(' '), write(C), nl,
    write(D), write(' '), write(E), write(' '), write(F), nl,
    write(G), write(' '), write(H), write(' '), write(I), nl,
    nl.
```

```
?- solve([1,2,3,4,5,6,7,0,8]).
Searching with depth limit: 1
Goal State Reached!
1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

true
```

Conclusion: Hence we solved the 8 Puzzle problem in prolog.

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 07=>

a. Write a program in prolog for Water jug problem

water.pl

```
% Represent state as state(Jug5, Jug3)
goal(state(4,_)).      % Goal: 4 liters in 5L jug
% Valid moves
move(state(X,Y), state(5,Y)) :- X < 5.
move(state(X,Y), state(X,3)) :- Y < 3.
move(state(X,Y), state(0,Y)) :- X > 0.
move(state(X,Y), state(X,0)) :- Y > 0.
move(state(X,Y), state(X1,Y1)) :-
    X > 0, Y < 3,
    Transfer is min(X, 3-Y),
    X1 is X - Transfer,
    Y1 is Y + Transfer.
move(state(X,Y), state(X1,Y1)) :-
    Y > 0, X < 5,
    Transfer is min(Y, 5-X),
    Y1 is Y - Transfer,
    X1 is X + Transfer.
% DFS to find solution
solve(Start) :-
    dfs(Start, []).
dfs(State, _) :-
    goal(State),
    write('Goal reached: '), write(State), nl, !.
dfs(State, Visited) :-
    write('Current state: '), write(State), nl,
    move(State, Next),
    \+ member(Next, Visited),
    dfs(Next, [State | Visited]).
```

```
?- solve(state(1,1)).
Current state: state(1,1)
Current state: state(5,1)
Current state: state(5,3)
Current state: state(0,3)
Current state: state(0,0)
Current state: state(5,0)
Current state: state(2,3)
Current state: state(2,0)
Current state: state(0,2)
Current state: state(5,2)
Goal reached: state(4,3)
true
```

Conclusion: Hence we solved water jug problem in prolog.

Name: Mayank Mandar Chandratre
Roll No. 07

Practical 08 =>

a. Write a program to implement a Tic-Tac-Toe game.

tic.pl

```prolog
play :-
    initial_board(Board),
    display_board(Board),
    write('Game Start! You are X.'), nl,
    play_turn(Board, x).

initial_board(['-','-','-','-','-','-','-','-','-']).

% --- 2. Display Board ---
display_board([A,B,C,D,E,F,G,H,I]) :-
    nl,
    write(' '), write(A), write(' | '), write(B), write(' | '),
    write('---+---+---'), nl,
    write(' '), write(D), write(' | '), write(E), write(' | '),
    write('---+---+---'), nl,
    write(' '), write(G), write(' | '), write(H), write(' | '),
    nl.

% --- 3. Game Logic ---

% Condition: X Wins
play_turn(Board, x) :-
    win(x, Board),
    write('X Wins!'), nl.

% Condition: O Wins
play_turn(Board, o) :-
    win(o, Board),
    write('O Wins!'), nl.

% Condition: Draw
play_turn(Board, _) :-
    \+ member('-', Board),
    write('It is a Draw!'), nl.

% Turn: X (Human Player)
play_turn(Board, x) :-
    write('Enter move (1-9): '),
    read(Pos),
    (valid_move(Board, Pos) ->
        update_board(Board, Pos, x, NewBoard),
        display_board(NewBoard),
        play_turn(NewBoard, o)
    ;
        write('Invalid move! Try again.'), nl,
```

tic.pl

```prolog
play_turn(Board, x) :-
    ;
    ).

% Turn: O (Computer - Simple Random AI)
play_turn(Board, o) :-
    write('Computer (O) is thinking...'), nl,
    move_computer(Board, NewBoard),
    display_board(NewBoard),
    play_turn(NewBoard, x).

% --- 4. Helpers ---

% Check if move is valid (spot is empty '-')
valid_move(Board, Pos) :-
    nth1(Pos, Board, '-').

% Update the list with the move
update_board([_|T], 1, Player, [Player|T]).
update_board([H|T], Pos, Player, [H|Rest]) :-
    Pos > 1,
    Pos1 is Pos - 1,
    update_board(T, Pos1, Player, Rest).

% Computer Logic: Pick the first available empty spot
move_computer(Board, NewBoard) :-
    nth1(Pos, Board, '-'),
    update_board(Board, Pos, o, NewBoard), !.

% --- 5. Winning Conditions ---
% Rows
win(P, [P,P,P,_,_,_,_,_,_]).
win(P, [_,_,_,P,P,P,_,_,_]).
win(P, [_,_,_,_,_,_,P,P,P]).
% Columns
win(P, [P,_,_,P,_,_,P,_,_]).
win(P, [_,P,_,_,P,_,_,P,_]).
win(P, [_,_,P,_,_,P,_,_,P]).
% Diagonals
win(P, [P,_,_,_,P,_,_,_,P]).
win(P, [_,_,P,_,P,_,P,_,_]).
```

Name: Mayank Mandar Chandratre
Roll No. 07

```
 - | - | -
---+---+---
 - | - | -
---+---+---
 - | - | -

Game Start! You are X.
Enter move (1-9): 5
|: .

 - | - | -
---+---+---
 - | x | -
---+---+---
 - | - | -

Computer (O) is thinking...

 o | - | -
---+---+---
 - | x | -
---+---+---
 - | - | -

Enter move (1-9): |: 3.

 o | - | x
---+---+---
 - | x | -
---+---+---
 - | - | -

Computer (O) is thinking...

 o | o | x
---+---+---
 - | x | -
---+---+---
 - | - | -

Enter move (1-9): |: 7.

 o | o | x
---+---+---
 - | x | -
---+---+---
 x | - | -

Computer (O) is thinking...

 o | o | x
---+---+---
 o | x | -
---+---+---
 x | - | -

X Wins!
true .
```

Conclusion: Hence we made a Tic-Tac-Toe Game in Prolog.

Name: Mayank Mandar Chandratre
Roll No. 07