

Practical 01 =>
Prolog installation.

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

The image consists of three screenshots of the SWI-Prolog website. The first screenshot shows the homepage with the SWI-Prolog logo, a navigation menu, and a description of the software. The second screenshot shows the 'SWI-Prolog downloads' page, which lists the development and stable releases, and provides links to download channels. The third screenshot shows the 'Download binary' page, which includes a warning about Windows antivirus software and a checkbox to enable the actual download link. A 'Recent download history' popup is visible in the bottom right corner of the third screenshot.

SWI-Prolog
Robust, mature, free. **Prolog for the real world.**

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY COMMERCIAL WIKI

SWI-Prolog
Sources/building
Docker images
Add-ons
Browse GIT

SWI-Prolog is a comprehensive free Prolog environment. Since its start in 1987, SWI-Prolog has been driven by the needs of real world applications. SWI-Prolog is widely used in research and education as well as commercial applications. Join over a million users who have downloaded SWI-Prolog. [more...](#)

Download SWI-Prolog Get Started Try SWI-Prolog online (SWISH)
Run SWI-Tinker in your browser

Star 1,147 Sponsor

Did you know? julian pack offers match based dates Search Documentation:

SWI-Prolog downloads

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY COMMERCIAL WIKI

- Development release ★
- Stable release
- Daily builds for Windows
- Browse GIT repository

Available download channels

The **development** version is released roughly every two to four weeks. It is typically robust, provides the latest features and possible issues are resolved quickly. This is the recommended version for most users.

The **stable** release is infrequently updated. After an update, the stable version receives only critical patches that are extremely unlikely to break source or binary compatibility. The stable versions are intended for deploying systems or other use cases that require a fully predictable installation.

Did you know? SWI-Prolog has an active **Discourse forum** Search Documentation:

Download binary

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY COMMERCIAL WIKI

⚠ Windows antivirus software works using *signatures* and *heuristics*. Using the huge amount of viruses and malware known today, arbitrary executables are often **falsely classified as malicious**. [Google Safe Browsing](#), used by most modern browsers, therefore often classifies our Windows binaries as malware. You can use e.g., [VirusTotal](#) to verify files with a large number of antivirus programs.

Our Windows binaries are cross-compiled on an isolated Linux container. The integrity of the binaries on the server is regularly verified by validating its SHA256 fingerprint.

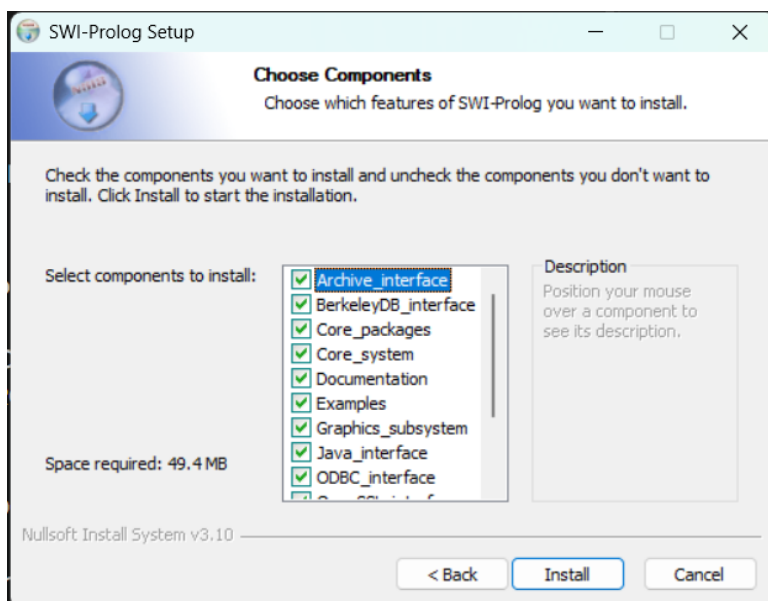
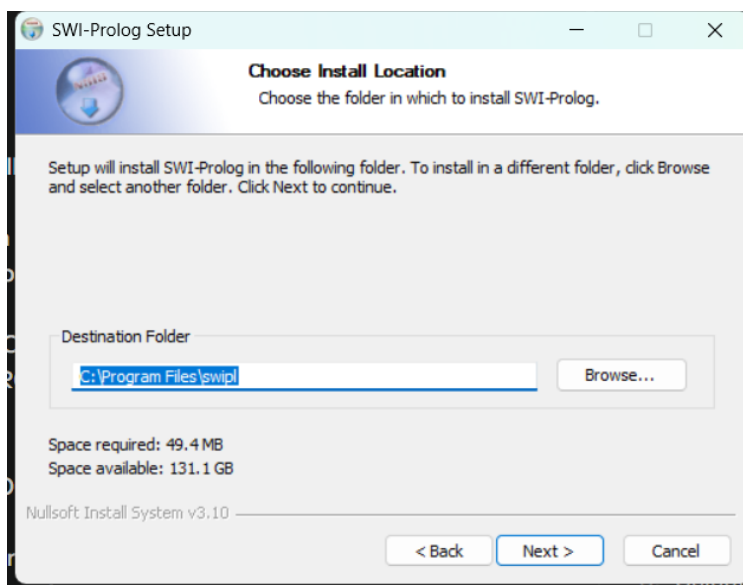
Please select the checkbox below to enable the actual download link.

☒ I understand
[Download swipl-9.2.9-1.x64.exe](#) (SHA256: 0e6dbf5f4bb245344a25f72715f5d793d1787bdee9ea1735ccb67b35f1e037c)
[VIRUSTOTAL Scan Result](#)

Recent download history
swipl-9.2.9-1.x64.exe
13.7 MB • 4 minutes ago
Full download history

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
1  cat(tom).
2  loves_to_eat(kunal,pasta).
3  of_color(hair,black).
4  loves_to_play_games(nawaz).
5  lazy(pratyusha).
6  like(ram,mango).
7  girl(seema).
8  red(rose).
9  likes(bill,gold).
10 owns(john,gold).
11
12 |
```

```
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
1  symptom(chicken_pox,high_fever).
2  symptom(chicken_pox,chills).
3  symptom(flu,chills).
4  symptom(cold,mild_body_ache).
5  symptom(flu,severe_body_ache).
6  symptom(cold,runny_nose).
7  symptom(flu,runny_nose).
8  symptom(flu,moderate_cough).
9  girl(priya).
10 girl(tiyasha).
11 girl(jaya).
12 can_cook(priya).
13 can_cook(jaya).
14 can_cook(tiyasha).
15 likes(priya, jaya) :- can_cook(jaya).
16 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.

?-
```

3)

 program3.pl

```
1  sing_a_song(ananya).
2  listens_to_music(rohit).
3  listens_to_music(ananya):-sing_a_song(ananya).
4  happy(ananya):-sing_a_song(ananya).
5  happy(rohit):-listens_to_music(rohit).
6  plays_guitar(rohit):-listens_to_music(rohit).
7  dog(rottweiler).
8  cat(sphynx).
9  dog(poodle).
10 dog(bulldog).
11 cat(bengali).
12 dog(dobernmann).
13 cat(himalayan).
14 cat(singapura).
15
```

```
Warning: Use :- discontinuous 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: 1 (h for help)
Action? ,

?-
|   ■
```

4)

program4.pl

```
1  in(room,banana).
2  at(ceiling,banana).
3  strong(monkey).
4  grasp(monkey).
5  climb(monkey,chair).
6  push(monkey,chair):-
7  strong(monkey).
8  under(banana,chair):-
9  push(monkey,chair).
10 canreach(banana,monkey):-
11 at(floor,banana);
12 at(ceiling,banana),
13 under(banana,chair),
14 climb(monkey,chair).
15 canget(banana,monkey):-
16 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.
```

```
?-
```

5)

 program5.pl

```
1 factorial(0, 1) :- !.  
2 factorial(N, F) :- N > 0,  
3     |         |         |         |     N1 is N - 1,  
4     |         |         |         |     factorial(N1, F1),  
5     |         |         |         |     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.
```

```
?-
```

Practical 03 =>
Arithmetic Programs in Prolog.
1)

```

1  % Check if a number is prime
2  is_prime(2) :- !.
3  is_prime(N) :-
4      N > 2,
5      N mod 2 \= 0,
6      \+ has_odd_divisor(N, 3).
7
8  has_odd_divisor(N, D) :-
9      D * D <= N,
10     (N mod D =:= 0 ; D1 is D + 2, has_odd_divisor(N, D1)).
11
12 % Find all primes up to N
13 primes_up_to(N, Primes) :-
14     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.
?-
```


2)

```
arith2.pl
1  % Calculate GCD using Euclidean algorithm
2  gcd(A, 0, A) :- !.
3  gcd(A, B, G) :-
4      B > 0,
5      R is A mod B,
6      gcd(B, R, G).
7
8  % Calculate LCM
9  lcm(A, B, L) :-
10     gcd(A, B, G),
11     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.
```

```
?- ■
```

3)

```
arith3.pl
1  % Program: Count to 10
2
3  % Base case: When the number is 10, print it and stop.
4  count_to_10(10) :-
5      write(10), nl.
6
7  % Recursive step: Print X, calculate Y = X + 1, then count(Y).
8  count_to_10(X) :-
9      write(X), nl,
10     Y is X + 1,
11     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 .
```

```
?-
|   █
```

4)

 arith4.pl

```
1 % Calculator predicate
2 calc(Operator, X, Y, Result) :-
3     Operator = '+', Result is X + Y;
4     Operator = '-', Result is X - Y;
5     Operator = '*', Result is X * Y;
6     Operator = '/', Y \= 0, Result is X / Y;
7     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.
```

```
?-
```

5)

```
arith5.pl
1  % Fibonacci number
2  fibonacci(0, 0) :- !.
3  fibonacci(1, 1) :- !.
4  fibonacci(N, F) :-
5      N > 1,
6      N1 is N - 1,
7      N2 is N - 2,
8      fibonacci(N1, F1),
9      fibonacci(N2, F2),
10     F is F1 + F2.
11
12 % Print first N fibonacci numbers
13 print_fibonacci(0) :- !.
14 print_fibonacci(N) :-
15     N > 0,
16     N1 is N - 1,
17     print_fibonacci(N1),
18     fibonacci(N1, F),
19     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.

?- ■
```

Practical 04 =>

a. Program to generate Family tree using PROLOG

```
family_tree.pl

17 parent(mandar, mayank).
18 parent(mandar, mihir).
19 parent(malvika, mayank).
20 parent(malvika, mihir).
21
22 % --- Rules (Logic) from Assignment 4 ---
23
24 % X is a grandparent of Y if X is a parent of Z, and Z is a parent of Y
25 grandparent(X, Y) :-
26     parent(X, Z),
27     parent(Z, Y).
28
29 % X and Y are siblings if they share a parent Z and X is not Y
30 sibling(X, Y) :-
31     parent(Z, X),
32     parent(Z, Y),
33     X \= Y.
34
35 % X is the father of Y if X is a parent of Y and X is male
36 father(X, Y) :-
37     parent(X, Y),
38     male(X).
39
40 % X is the mother of Y if X is a parent of Y and X is female
41 mother(X, Y) :-
42     parent(X, Y),
43     female(X).
44
45 % X is an ancestor of Y (Recursive definition)
46 ancestor(X, Y) :- parent(X, Y).
47 ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
48
49 % X is a descendant of Y
50 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 .

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

?- descendant(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.

Practical 05 =>

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

```
pr5.pl
2  initial_state(state(at_door, at_door, onfloor, hanging)).
3  goal_state(state(_, _, _, grasped)).
4
5  % --- Move Rules ---
6
7  % 1. Walk from Door to Banana
8  move(state(at_door, CrateLoc, onfloor, Banana),
9      | state(at_banana, CrateLoc, onfloor, Banana), 'Walk to Banana').
10
11 % 2. Walk from Banana to Door
12 move(state(at_banana, CrateLoc, onfloor, Banana),
13     | state(at_door, CrateLoc, onfloor, Banana), 'Walk to Door').
14
15 % 3. Push Crate from Door to Banana (FIXED: Monkey moves WITH crate)
16 move(state(at_door, at_door, onfloor, Banana),
17     | state(at_banana, at_banana, onfloor, Banana), 'Push Crate to Banana').
18
19 % 4. Push Crate from Banana to Door (FIXED: Monkey moves WITH crate)
20 move(state(at_banana, at_banana, onfloor, Banana),
21     | state(at_door, at_door, onfloor, Banana), 'Push Crate to Door').
22
23 % 5. Climb Crate
24 move(state(at_banana, at_banana, onfloor, Banana),
25     | state(at_banana, at_banana, oncrate, Banana), 'Climb Crate').
26
27 % 6. Grasp Banana
28 move(state(at_banana, at_banana, oncrate, hanging),
29     | state(at_banana, at_banana, oncrate, grasped), 'Grasp Banana').
30
31 % --- Solver (FIXED with Cycle Detection) ---
32
33 % Entry point for the user
34 solve_problem :-
35     initial_state(Start),
36     solve(Start, [], Path), % Start with empty visited list
37     write('Solution Path:'), nl,
38     print_path(Path).
39
40 % Base Case: Goal reached
41 solve(State, _, []) :-
42     goal_state(State).
43
44 % Recursive Step: Try a move
45 solve(CurrentState, Visited, [Action | Path]) :-
46     move(CurrentState, NextState, Action),
47     \+ member(NextState, Visited),
48     solve(NextState, [NextState | Visited], Path).
49
50 % Helper to print the path neatly
51 print_path([]).
52 print_path([Action | Rest]) :-
53     write(' -> '), write(Action), nl,
54     print_path(Rest).
```

```
% 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
1 hanoi(1, Source, Target, _):-
2   write('Move disk 1 from '), write(Source), write(' to '), write(Target), nl.
3
4 hanoi(N, Source, Target, Auxiliary):-
5   N > 1,
6   M is N - 1,
7   hanoi(M, Source, Auxiliary, Target),
8   write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Target), nl,
9   hanoi(M, Auxiliary, Target, Source).
```

```
hanoi.pl
1 hanoi(1, Source, Target, _):-
2   write('Move disk 1 from '), write(Source), write(' to '), write(Target), nl.
3
4 hanoi(N, Source, Target, Auxiliary):-
5   N > 1,
6   M is N - 1,
7   hanoi(M, Source, Auxiliary, Target),
8   write('Move disk '), write(N), write(' from '), write(Source), write(' to '), write(Target), nl,
9   hanoi(M, Auxiliary, Target, Source).
```

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

Practical 06 =>

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

```
puzzle.pl
1 % =====
2 % 8-Puzzle with Depth Limit (Prevents Loops)
3 % =====
4
5 goal([1,2,3,
6     4,5,6,
7     7,8,0]).
8
9 % --- List Helpers ---
10 set_elem([_|T], 0, X, [X|T]).
11 set_elem([H|T], I, X, [H|R]) :-
12     I > 0, I1 is I - 1, set_elem(T, I1, X, R).
13
14 swap(List, I, J, Result) :-
15     nth0(I, List, A), nth0(J, List, B),
16     set_elem(List, I, B, Temp),
17     set_elem(Temp, J, A, Result).
18
19 % --- Move Logic ---
20 move(State, NextState) :-
21     nth0(Blank, State, 0),
22     move_blank(Blank, NewBlank),
23     swap(State, Blank, NewBlank, NextState).
24
25 move_blank(0,1). move_blank(0,3).
26 move_blank(1,0). move_blank(1,2). move_blank(1,4).
27 move_blank(2,1). move_blank(2,5).
28 move_blank(3,0). move_blank(3,4). move_blank(3,6).
29 move_blank(4,1). move_blank(4,3). move_blank(4,5). move_blank(4,7).
30 move_blank(5,2). move_blank(5,4). move_blank(5,8).
31 move_blank(6,3). move_blank(6,7).
32 move_blank(7,4). move_blank(7,6). move_blank(7,8).
33 move_blank(8,5). move_blank(8,7).
34
35 % --- SOLVER (Iterative Deepening) ---
36
37 solve(Start) :-
38     % Try to find path of length 1, then 2, then 3... up to 15
39     between(1, 15, Limit),
40     write('Searching with depth limit: '), write(Limit), nl,
41     path(Start, [Start], Limit).
42
43 path(State, Visited, _) :-
44     goal(State),
45     write('Goal State Reached!'), 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.

Practical 07=>

- a. Write a program in prolog for Water jug problem

```
water.pl
1 % Represent state as state(Jug5, Jug3)
2 goal(state(4,_)). % Goal: 4 liters in 5L jug
3 % Valid moves
4 move(state(X,Y), state(5,Y)) :- X < 5.
5 move(state(X,Y), state(X,3)) :- Y < 3.
6 move(state(X,Y), state(0,Y)) :- X > 0.
7 move(state(X,Y), state(X,0)) :- Y > 0.
8 move(state(X,Y), state(X1,Y1)) :-
9     X > 0, Y < 3,
10    Transfer is min(X, 3-Y),
11    X1 is X - Transfer,
12    Y1 is Y + Transfer.
13 move(state(X,Y), state(X1,Y1)) :-
14    Y > 0, X < 5,
15    Transfer is min(Y, 5-X),
16    Y1 is Y - Transfer,
17    X1 is X + Transfer.
18 % DFS to find solution
19 solve(Start) :-
20     dfs(Start, []).
21 dfs(State, _) :-
22     goal(State),
23     write('Goal reached: '), write(State), nl, !.
24 dfs(State, Visited) :-
25     write('Current state: '), write(State), nl,
26     move(State, Next),
27     \+ member(Next, Visited),
28     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.

Practical 08 =>

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

```
ticpl
4 play :-
5     initial_board(Board),
6     display_board(Board),
7     write('Game Start! You are X. '), nl,
8     play_turn(Board, x).
9
10 initial_board(['-', '-', '-', '-', '-', '-', '-', '-', '-']).
11
12 % --- 2. Display Board ---
13 display_board([A,B,C,D,E,F,G,H,I]) :-
14     nl,
15     write(' '), write(A), write(' | '), write(B), write(' | '),
16     write('-----'), nl,
17     write(' '), write(D), write(' | '), write(E), write(' | '),
18     write('-----'), nl,
19     write(' '), write(G), write(' | '), write(H), write(' | '),
20     nl.
21
22 % --- 3. Game Logic ---
23
24 % Condition: X Wins
25 play_turn(Board, x) :-
26     win(x, Board),
27     write('X Wins!'), nl.
28
29 % Condition: O Wins
30 play_turn(Board, o) :-
31     win(o, Board),
32     write('O Wins!'), nl.
33
34 % Condition: Draw
35 play_turn(Board, _) :-
36     \+ member('-', Board),
37     write('It is a Draw!'), nl.
38
39 % Turn: X (Human Player)
40 play_turn(Board, x) :-
41     write('Enter move (1-9): '),
42     read(Pos),
43     (valid_move(Board, Pos) ->
44         update_board(Board, Pos, x, NewBoard),
45         display_board(NewBoard),
46         play_turn(NewBoard, o)
47     );
48     write('Invalid move! Try again. '), nl,
```

```
ticpl
40 play_turn(Board, x) :-
41     ;
42
43
44 % Turn: O (Computer - Simple Random AI)
45 play_turn(Board, o) :-
46     write('Computer (O) is thinking... '), nl,
47     move_computer(Board, NewBoard),
48     display_board(NewBoard),
49     play_turn(NewBoard, x).
50
51 % --- 4. Helpers ---
52
53 % Check if move is valid (spot is empty '-')
54 valid_move(Board, Pos) :-
55     nth1(Pos, Board, '-').
56
57 % Update the list with the move
58 update_board([_|T], 1, Player, [Player|T]).
59 update_board([H|T], Pos, Player, [H|Rest]) :-
60     Pos > 1,
61     Pos1 is Pos - 1,
62     update_board(T, Pos1, Player, Rest).
63
64 % Computer Logic: Pick the first available empty spot
65 move_computer(Board, NewBoard) :-
66     nth1(Pos, Board, '-'),
67     update_board(Board, Pos, o, NewBoard), !.
68
69 % --- 5. Winning Conditions ---
70
71 % Rows
72 win(P, [P,P,P,_,_,_,_,_,_]).
73 win(P, [_P,P,P,_,_,_,_,_]).
74 win(P, [_,_P,P,P,_,_,_,_]).
75
76 % Columns
77 win(P, [_,_,_,P,_,_,_,P]).
78 win(P, [_,_,_,_,P,_,_,P]).
79 win(P, [_,_,_,_,_,P,_,P]).
80
81 % Diagonals
82 win(P, [P,_,_,P,_,_,_,P]).
83 win(P, [_,P,_,_,P,_,_,_]).
```

```

- | - | -
-+--+
- | - | -
-+--+
- | - | -

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.