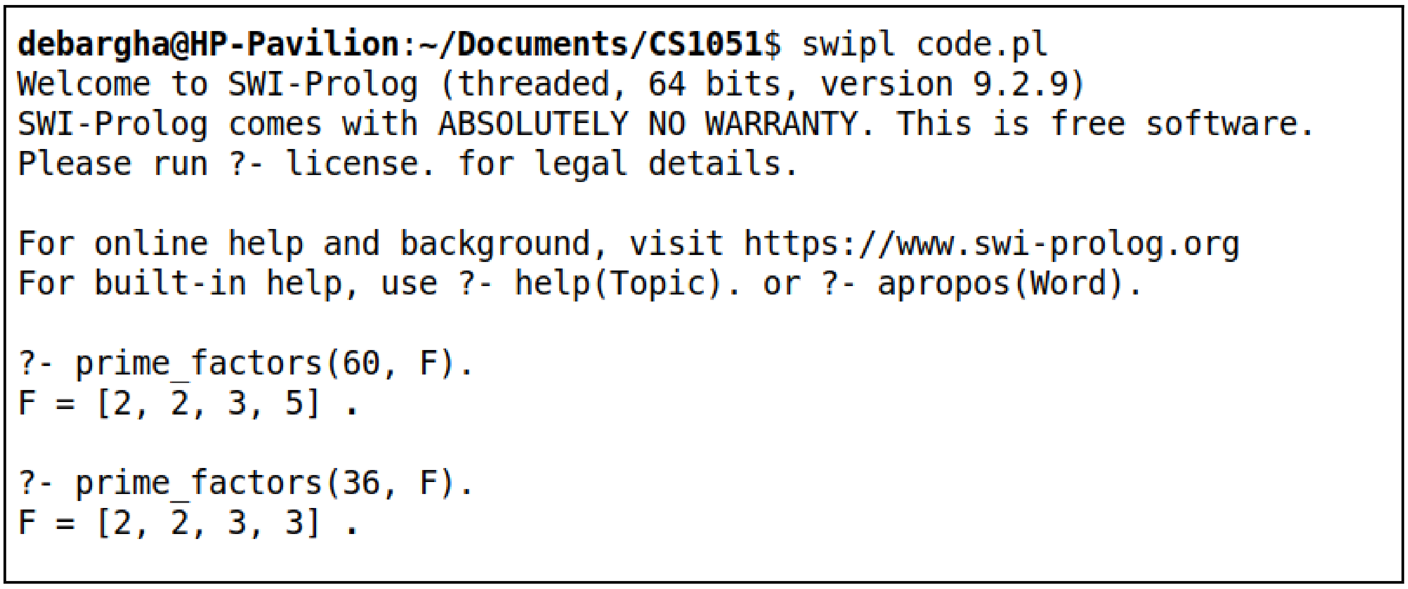
Assignment-4

1. Write a Prolog program to implement prime factors of a given (user input) number.

code.pl

```
% Check divisibility
divides(X, Y) :- 0 is Y mod X.

% Main predicate
prime_factors(N, Factors) :-
    N > 1,
    factor(N, 2, Factors).

% Base case
factor(1, _, []).

% If F divides N, include it and divide again
factor(N, F, [F | Rest]) :-
    divides(F, N),
    N1 is N // F,
    factor(N1, F, Rest).

% If F does not divide N, try next F
factor(N, F, Factors) :-
    F * F =< N,
    next_factor(F, NF),
    factor(N, NF, Factors).

% If F * F > N, remaining N is prime
factor(N, F, [N]) :-
    F * F > N.

% Increment factor (skip even numbers after 2)
next_factor(2, 3) :- !.
next_factor(F, NF) :-
    NF is F + 2.
```

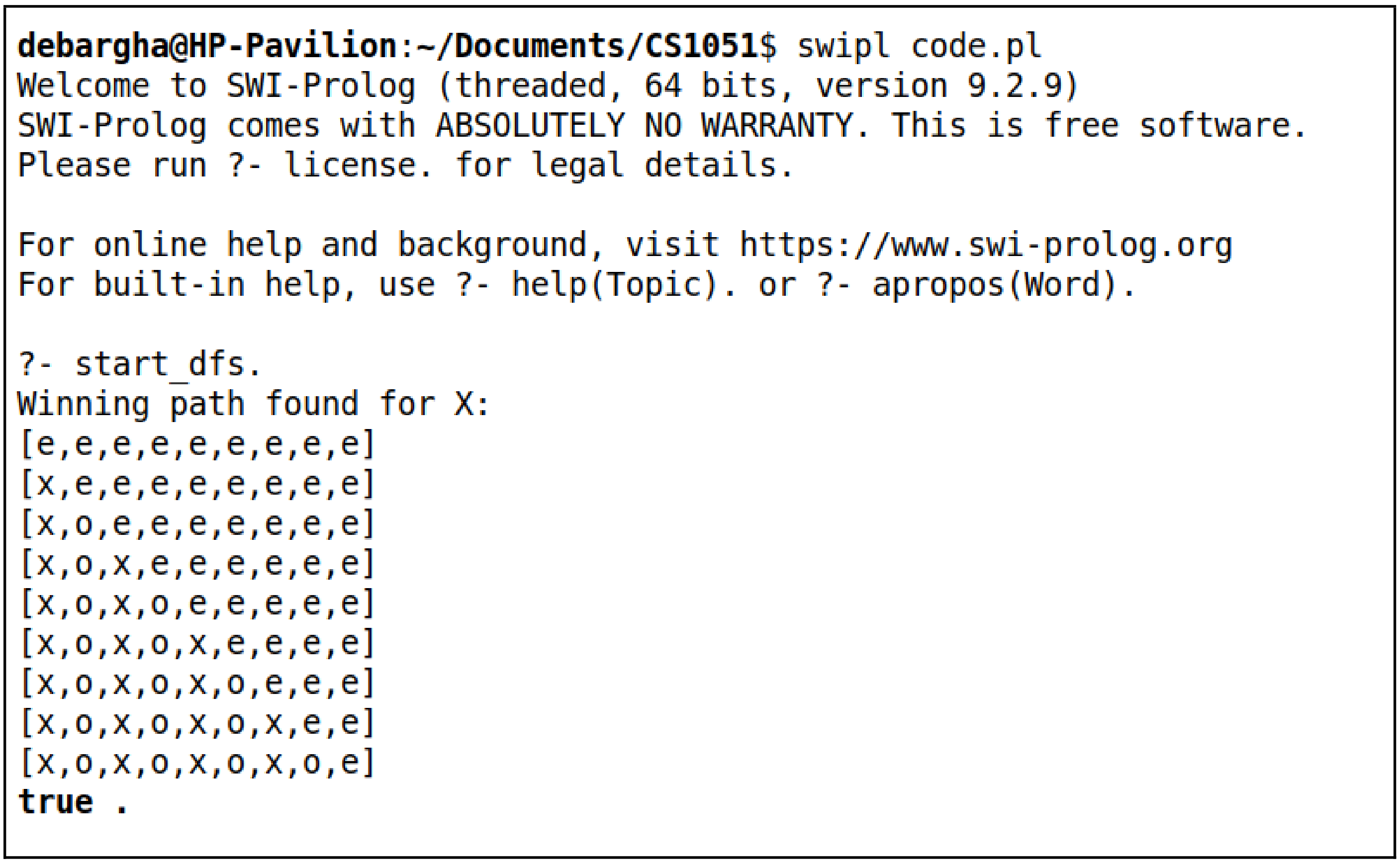


```
debargha@HP-Pavilion:~/Documents/CS1051$ swipl code.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- prime_factors(60, F).
F = [2, 2, 3, 5] .

?- prime_factors(36, F).
F = [2, 2, 3, 3] .
```

2. Write a Prolog program to implement Depth First Search (DFS) for Tic-Tac-Toe.

code.pl

```
% Switch player
switch(x, o).
switch(o, x).

% Check winning condition
win(Board, Player) :-
    ( Board = [Player, Player, Player, _, _, _, _, _, _] % Row 1
    ; Board = [_, _, _, Player, Player, Player, _, _, _] % Row 2
    ; Board = [_, _, _, _, _, _, Player, Player, Player] % Row 3
    ; Board = [Player, _, _, Player, _, _, Player, _, _] % Column 1
    ; Board = [_, Player, _, _, Player, _, _, Player, _] % Column 2
    ; Board = [_, _, Player, _, _, Player, _, _, Player] % Column 3
    ; Board = [Player, _, _, _, Player, _, _, _, Player] % Diagonal 1
    ; Board = [_, _, Player, _, Player, _, Player, _, _] % Diagonal 2
    ).

% Check if board is full
full(Board) :- \+ member(e, Board).

% Make a move
move(Board, Player, NewBoard) :-
    nth0(Pos, Board, e), % Find an empty cell
    replace(Board, Pos, Player, NewBoard).

% Replace an element at position
replace([_|T], 0, X, [X|T]).
replace([H|T], I, X, [H|R]) :-
    I > 0,
    I1 is I - 1,
    replace(T, I1, X, R).

% DFS Search
dfs(Board, Player, Path) :-
    dfs_util(Board, Player, [Board], Path).

dfs_util(Board, Player, _Visited, [Board]) :-
    win(Board, Player), !. % If win found, stop

dfs_util(Board, Player, Visited, [Board | Path]) :-
    move(Board, Player, NewBoard),
    \+ member(NewBoard, Visited), % Avoid revisiting
    switch(Player, NextPlayer),
    dfs_util(NewBoard, NextPlayer, [NewBoard | Visited], Path).

% Start DFS Search
start_dfs :-
    Start = [e,e,e,e,e,e,e,e,e],
    dfs(Start, x, Path),
    write('Winning path found for X: '), nl,
    print_path(Path).

% Print Path
print_path([]).
print_path([Board | Rest]) :-
    write(Board), nl,
    print_path(Rest).
```



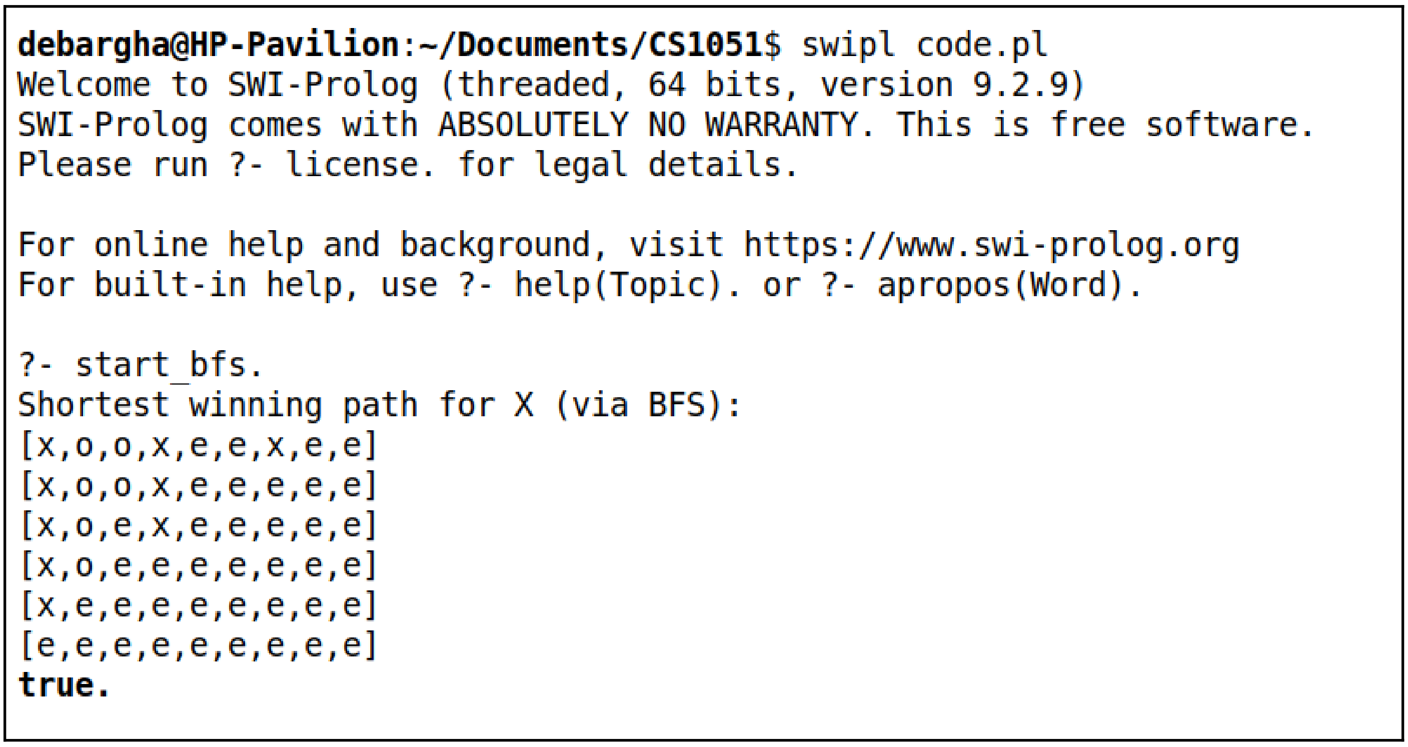
```
debargha@HP-Pavilion:~/Documents/CS1051$ swipl code.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

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

?- start_dfs.
Winning path found for X:
[e,e,e,e,e,e,e,e,e]
[x,e,e,e,e,e,e,e,e]
[x,o,e,e,e,e,e,e,e]
[x,o,x,e,e,e,e,e,e]
[x,o,x,o,e,e,e,e,e]
[x,o,x,o,x,e,e,e,e]
[x,o,x,o,x,o,e,e,e]
[x,o,x,o,x,o,x,e,e]
[x,o,x,o,x,o,x,o,e]
true .
```

3. Write a Prolog program to implement Breadth First Search for Tic-Tac-Toe.

code.pl

```
% Switch Player
switch(x, o).
switch(o, x).

% Check Winning Condition
win(Board, Player) :-
    ( Board = [Player, Player, Player, _, _, _, _, _, _] % Row 1
    ; Board = [_, _, _, Player, Player, Player, _, _, _] % Row 2
    ; Board = [_, _, _, _, _, _, Player, Player, Player] % Row 3
    ; Board = [Player, _, _, Player, _, _, Player, _, _] % Column 1
    ; Board = [_, Player, _, _, Player, _, _, Player, _] % Column 2
    ; Board = [_, _, Player, _, _, Player, _, _, Player] % Column 3
    ; Board = [Player, _, _, _, Player, _, _, _, Player] % Diagonal 1
    ; Board = [_, _, Player, _, Player, _, Player, _, _] % Diagonal 2
    ).

% Replace Nth Element
replace([_|T], 0, X, [X|T]).
replace([H|T], I, X, [H|R]) :-
    I > 0,
    I1 is I - 1,
    replace(T, I1, X, R).

% Place Player in an empty cell 'e'
move(Board, Player, NewBoard) :-
    nth0(Pos, Board, e), % find an empty position
    replace(Board, Pos, Player, NewBoard).

% returns SolutionPath as a list
bfs(StartBoard, StartPlayer, SolutionPath) :-
    bfs_queue([ state(StartBoard, [StartBoard], StartPlayer) ], [], RevPath),
    reverse(RevPath, SolutionPath).

% Queue empty -> no solution
bfs_queue([], _, _) :-
    !, fail.

% If the current board is terminal
bfs_queue([ state(Board, Path, Player) | _Rest ], _Visited, Path) :-
    switch(Player, PrevPlayer), % PrevPlayer is the one who moved to get Board
    win(Board, PrevPlayer), !.

% Expand current state: generate moves, add them to end of queue (BFS)
bfs_queue([ state(Board, Path, Player) | RestQueue ], Visited, ResultPath) :-
    findall(state(NewBoard, [NewBoard | Path], NextPlayer),
```

```prolog
            ( move(Board, Player, NewBoard),
              \+ member(NewBoard, Visited), % not visited previously
              \+ member(NewBoard, Path), % avoid cycles in current path
              switch(Player, NextPlayer) % NextPlayer gets the turn
            ),
            NewStates),
    append(RestQueue, NewStates, UpdatedQueue),
    bfs_queue(UpdatedQueue, [Board | Visited], ResultPath).

% Helper to start a search from the empty board
start_bfs :-
    Start = [e,e,e,e,e,e,e,e,e],
    ( bfs(Start, x, Path) ->
        reverse(Path, OrderedPath),
        writeln('Shortest winning path for X (via BFS):'),
        print_path(OrderedPath)
    ; writeln('No winning path found (BFS exhausted).')
    ).

% Print Path (one board per line)
print_path([]).
print_path([Board | Rest]) :-
    write(Board), nl,
    print_path(Rest).
```



```
debargha@HP-Pavilion:~/Documents/CS1051$ swipl code.pl
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).

?- start_bfs.
Shortest winning path for X (via BFS):
[x,o,o,x,e,e,x,e,e]
[x,o,o,x,e,e,e,e,e]
[x,o,e,x,e,e,e,e,e]
[x,o,e,e,e,e,e,e,e]
[x,e,e,e,e,e,e,e,e]
[e,e,e,e,e,e,e,e,e]
true.
```