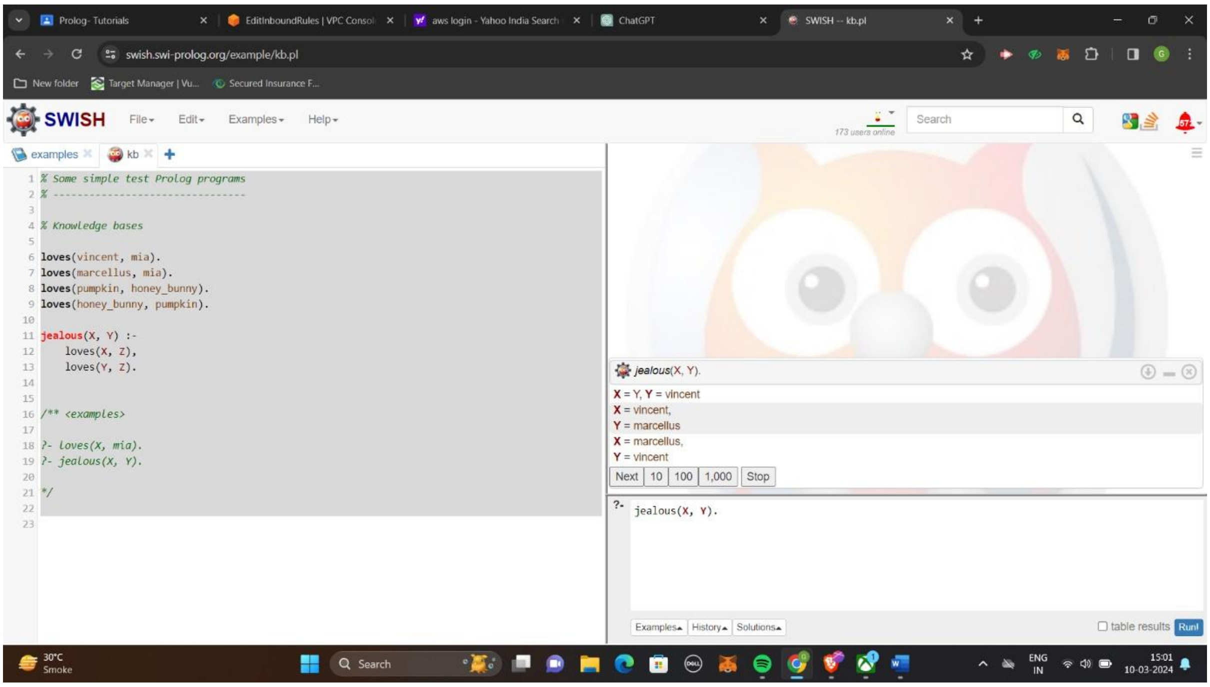
AI LAB – 8 9762-Aditi Gupta– Batch D

Prolog Programmes:

```
% Some simple test Prolog programs
% -------------------------------

% Knowledge bases

loves(vincent, mia). loves(marcellus,
mia). loves(pumpkin,
honey_bunny). loves(honey_bunny,
pumpkin).

jealous(X, Y) :-
loves(X, Z),    loves(Y,
Z).


/** <examples>

?- loves(X, mia).
?- jealous(X, Y).

*/
```



```
% Some simple test Prolog programs
% -------------------------------

% Knowledge bases

loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X, Y) :-
    loves(X, Z),
    loves(Y, Z).


/** <examples>

?- loves(X, mia).
?- jealous(X, Y).

*/
```

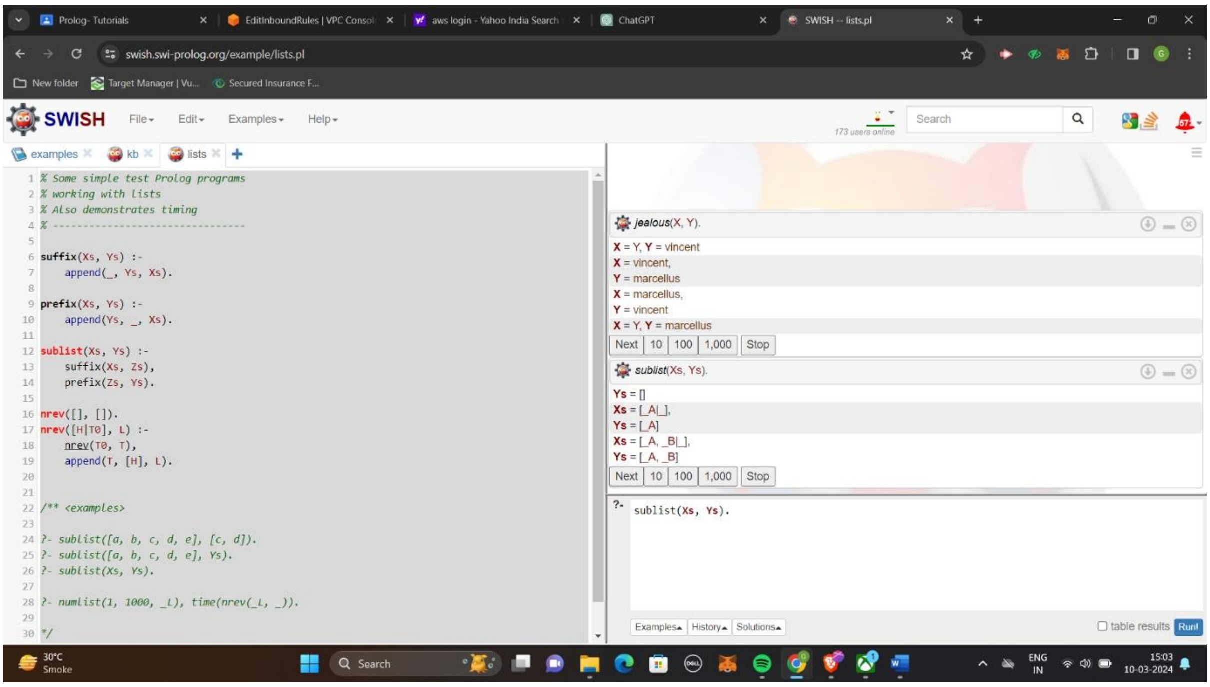
```
jealous(X, Y).
X = Y, Y = vincent
X = vincent,
Y = marcellus
X = marcellus,
Y = vincent
```

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

CODE 2:

```
% Some simple test Prolog programs

% working with lists

% Also demonstrates timing

% --------------------------------



suffix(Xs, Ys) :-

append(_, Ys, Xs).



prefix(Xs, Ys) :-

append(Ys, _, Xs).



sublist(Xs, Ys) :-

suffix(Xs, Zs),

prefix(Zs, Ys). nrev([], []).

nrev([H|T0], L) :-
```

```
        nrev(T0, T),
append(T, [H], L).



/** <examples>


?- sublist([a, b, c, d, e], [c, d]).
?- sublist([a, b, c, d, e], Ys).
?- sublist(Xs, Ys).


?- numlist(1, 1000, _L), time(nrev(_L, _)).


*/
```

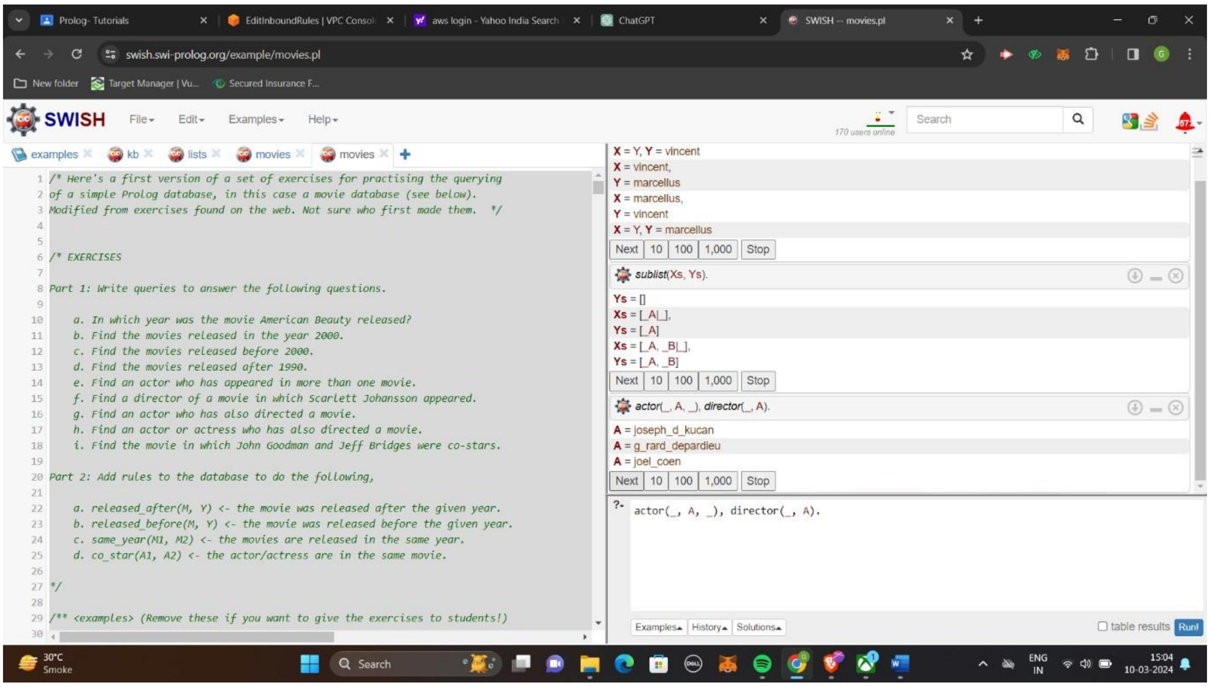


```
% Some simple test Prolog programs
% working with lists
% Also demonstrates timing
% -------------------------------

suffix(Xs, Ys) :-
    append(_, Ys, Xs).

prefix(Xs, Ys) :-
    append(Ys, _, Xs).

sublist(Xs, Ys) :-
    suffix(Xs, Zs),
    prefix(Zs, Ys).

nrev([], []).
nrev([H|T0], L) :-
    nrev(T0, T),
    append(T, [H], L).


/** <examples>

?- sublist([a, b, c, d, e], [c, d]).
?- sublist([a, b, c, d, e], Ys).
?- sublist(Xs, Ys).

?- numlist(1, 1000, _L), time(nrev(_L, _)).

*/
```

```
jealous(X, Y).
X = Y, Y = vincent
X = vincent,
Y = marcellus
X = marcellus,
Y = vincent
X = Y, Y = marcellus
```

```
sublist(Xs, Ys).
Ys = []
Xs = [_A|_],
Ys = [_A]
Xs = [_A, _B|_],
Ys = [_A, _B]
```

```
?- sublist(Xs, Ys).
```



```
/* Here's a first version of a set of exercises for practising the querying
of a simple Prolog database, in this case a movie database (see below).
Modified from exercises found on the web. Not sure who first made them.  */


/* EXERCISES

Part 1: Write queries to answer the following questions.

    a. In which year was the movie American Beauty released?
    b. Find the movies released in the year 2000.
    c. Find the movies released before 2000.
    d. Find the movies released after 1990.
    e. Find an actor who has appeared in more than one movie.
    f. Find a director of a movie in which Scarlett Johansson appeared.
    g. Find an actor who has also directed a movie.
    h. Find an actor or actress who has also directed a movie.
    i. Find the movie in which John Goodman and Jeff Bridges were co-stars.

Part 2: Add rules to the database to do the following,

    a. released_after(M, Y) <- the movie was released after the given year.
    b. released_before(M, Y) <- the movie was released before the given year.
    c. same_year(M1, M2) <- the movies are released in the same year.
    d. co_star(A1, A2) <- the actor/actress are in the same movie.

*/

/** <examples> (Remove these if you want to give the exercises to students!)
```

```
X = Y, Y = vincent
X = vincent,
Y = marcellus
X = marcellus,
Y = vincent
X = Y, Y = marcellus

sublist(Xs, Ys).
Ys = []
Xs = [_A|_],
Ys = [_A]
Xs = [_A, _B|_],
Ys = [_A, _B]

actor(_, A, _), director(_, A).
A = joseph_d_kucan
A = g_rard_depardieu
A = joel_coen

?- actor(_, A, _), director(_, A).
```

Code 5:

% A meta-interpreter implementing

% a tiny expert-system

% --------------------------------

prove(true) :- !.

prove((B, Bs)) :- !,

prove(B),

prove(Bs). prove(H)

:-   clause(H, B),

prove(B). prove(H) :-

askable(H),

writeln(H),

read(Answer),

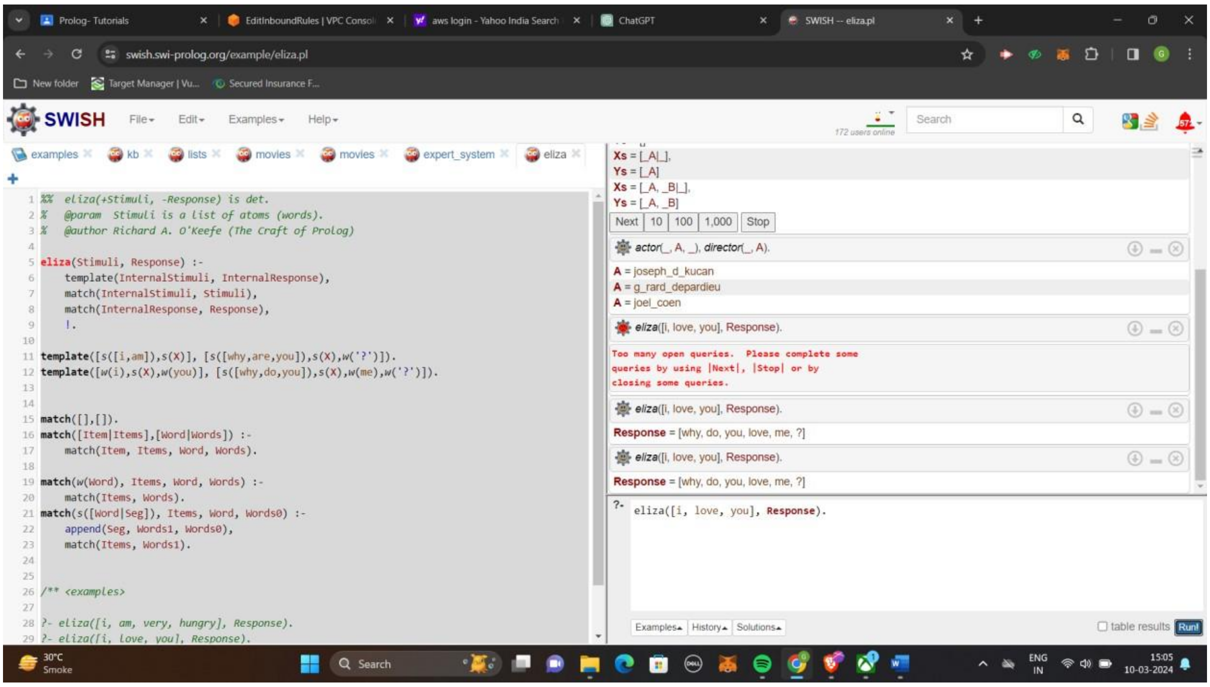
    Answer == yes.

```
good_pet(X) :- bird(X), small(X). good_pet(X)
:- cuddly(X), yellow(X).

bird(X) :- has_feathers(X), tweets(X).

yellow(tweety).

askable(tweets(_)).
askable(small(_)). askable(cuddly(_)).
askable(has_feathers(_)).


/** <examples>

?- prove(good_pet(tweety)).

*/
```

Code 6:

```
%%  eliza(+Stimuli, -Response) is det.
%   @param  Stimuli is a list of atoms (words). %   @author Richard A. O'Keefe
(The Craft of Prolog)

eliza(Stimuli, Response) :-
```

```
   template(InternalStimuli, InternalResponse),
match(InternalStimuli, Stimuli),    match(InternalResponse,
Response),
   !.

template([s([i,am]),s(X)], [s([why,are,you]),s(X),w('?')]).
template([w(i),s(X),w(you)], [s([why,do,you]),s(X),w(me),w('?')]).



match([],[]).
match([Item|Items],[Word|Words]) :-
match(Item, Items, Word, Words).

match(w(Word), Items, Word, Words) :-
match(Items, Words).
match(s([Word|Seg]), Items, Word, Words0) :-
append(Seg, Words1, Words0),    match(Items,
Words1).



/** <examples>

?- eliza([i, am, very, hungry], Response).
?- eliza([i, love, you], Response).

*/
```



```
%%  eliza(+Stimuli, -Response) is det.
%   @param  Stimuli is a list of atoms (words).
%   @author Richard A. O'Keefe (The Craft of Prolog)

eliza(Stimuli, Response) :-
    template(InternalStimuli, InternalResponse),
    match(InternalStimuli, Stimuli),
    match(InternalResponse, Response),
    !.

template([s([i,am]),s(X)], [s([why,are,you]),s(X),w('?')]).
template([w(i),s(X),w(you)], [s([why,do,you]),s(X),w(me),w('?')]).


match([],[]).
match([Item|Items],[Word|Words]) :-
    match(Item, Items, Word, Words).

match(w(Word), Items, Word, Words) :-
    match(Items, Words).
match(s([Word|Seg]), Items, Word, Words0) :-
    append(Seg, Words1, Words0),
    match(Items, Words1).


/** <examples>

?- eliza([i, am, very, hungry], Response).
?- eliza([i, love, you], Response).
```

```
Xs = [_A|_],
Ys = [_A]
Xs = [_A, _B|_],
Ys = [_A, _B]

actor(_, A, _), director(_, A).
A = joseph_d_kucan
A = g_rard_depardieu
A = joel_coen

eliza([i, love, you], Response).
Too many open queries.  Please complete some
queries by using |Next|, |Stop| or by
closing some queries.

eliza([i, love, you], Response).
Response = [why, do, you, love, me, ?]

eliza([i, love, you], Response).
Response = [why, do, you, love, me, ?]

?- eliza([i, love, you], Response).
```

Code :

```
% Render parse trees using a tree, but ignore lists Relies on native SVG

% support in the browser. IF THE ANSWER LOOKS EMPTY, COMMENT OR
REMOVE

% THE LINE BELOW.

:- use_rendering(svgtree, [list(false)]).


% A simple English DCG grammar

% ============================


s(s(NP,VP)) --> np(NP, Num), vp(VP, Num).


np(NP, Num) --> pn(NP, Num). np(np(Det,N), Num)

--> det(Det, Num), n(N, Num). np(np(Det,N,PP),

Num) --> det(Det, Num), n(N, Num), pp(PP).
```

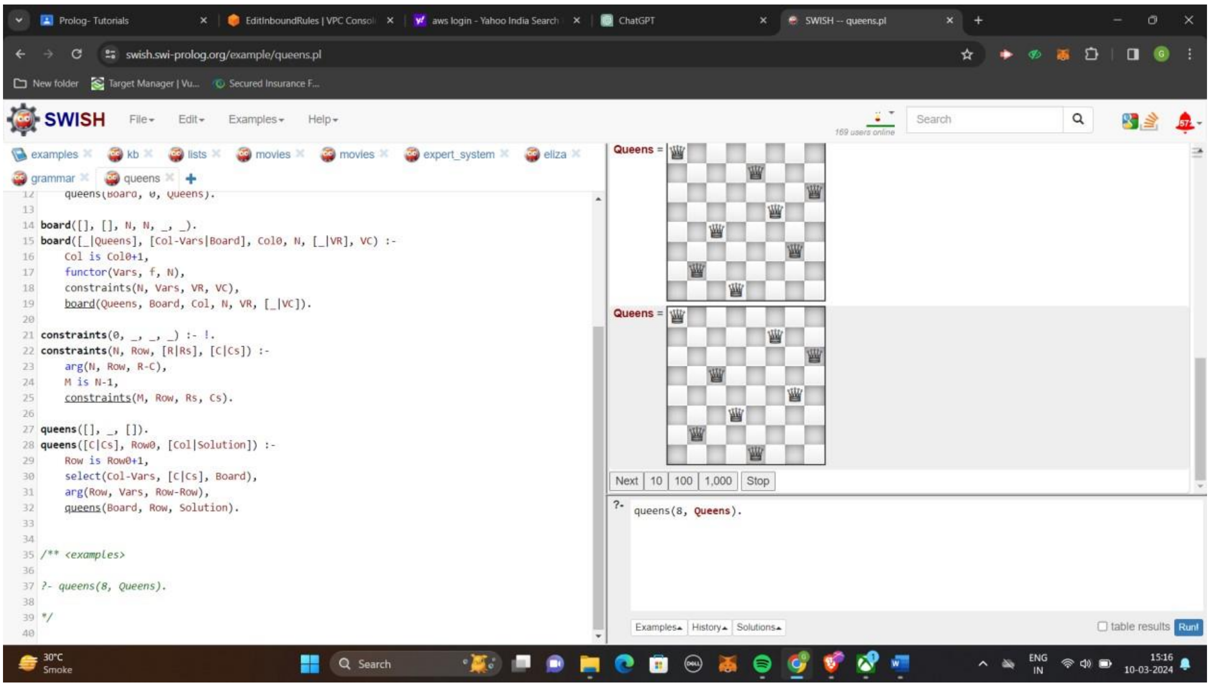
```
vp(vp(V,NP), Num) --> v(V, Num), np(NP, _). vp(vp(V,NP,PP),
Num) --> v(V, Num), np(NP, _), pp(PP).

pp(pp(P,NP)) --> p(P), np(NP, _).

det(det(a), sg) --> [a]. det(det(the),
_) --> [the].

pn(pn(john), sg) --> [john].

n(n(man), sg) --> [man]. n(n(men),
pl) --> [men]. n(n(telescope), sg) --
> [telescope].

v(v(sees), sg) --> [sees].
v(v(see), pl) --> [see]. v(v(saw),
_) --> [saw].

p(p(with)) --> [with].



/** <examples>

?- phrase(s(Tree), [john, saw, a, man, with, a, telescope]).
?- phrase(s(Tree), Sentence).
?- between(1, 8, N), length(S, N), phrase(s(_), S), writeln(S), sleep(0.2), false.
```

*/

Code:

```
% render solutions nicely.
:- use_rendering(chess).

%%    queens(+N, -Queens) is nondet.
%
%       @param       Queens is a list of column numbers for placing the queens. %
@author Richard A. O'Keefe (The Craft of Prolog)

queens(N, Queens) :-
length(Queens, N),
          board(Queens, Board, 0, N, _, _),
queens(Board, 0, Queens).

board([], [], N, N, _, _).
board([_|Queens], [Col-Vars|Board], Col0, N, [_|VR], VC) :-
Col is Col0+1,         functor(Vars, f, N),           constraints(N,
Vars, VR, VC),         board(Queens, Board, Col, N, VR,
[_|VC]).

constraints(0, _, _, _) :- !.
constraints(N, Row, [R|Rs], [C|Cs]) :-
arg(N, Row, R-C),    M is N-1,
          constraints(M, Row, Rs, Cs).
```

```
queens([], _, []). queens([C|Cs], Row0,
[Col|Solution]) :-
        Row is Row0+1,
        select(Col-Vars, [C|Cs], Board),
arg(Row, Vars, Row-Row),
queens(Board, Row, Solution).

/** <examples>

?- queens(8, Queens).

*/
```



```
    queens(Board, 0, Queens).

board([], [], N, N, _, _).
board([_|Queens], [Col-Vars|Board], Col0, N, [_|VR], VC) :-
    Col is Col0+1,
    functor(Vars, f, N),
    constraints(N, Vars, VR, VC),
    board(Queens, Board, Col, N, VR, [_|VC]).

constraints(0, _, _, _) :- !.
constraints(N, Row, [R|Rs], [C|Cs]) :-
    arg(N, Row, R-C),
    M is N-1,
    constraints(M, Row, Rs, Cs).

queens([], _, []).
queens([C|Cs], Row0, [Col|Solution]) :-
    Row is Row0+1,
    select(Col-Vars, [C|Cs], Board),
    arg(Row, Vars, Row-Row),
    queens(Board, Row, Solution).


/** <examples>

?- queens(8, Queens).

*/
```

```
?- queens(8, Queens).
```