1. **Write a prolog program to calculate the sum of two**

numbers.

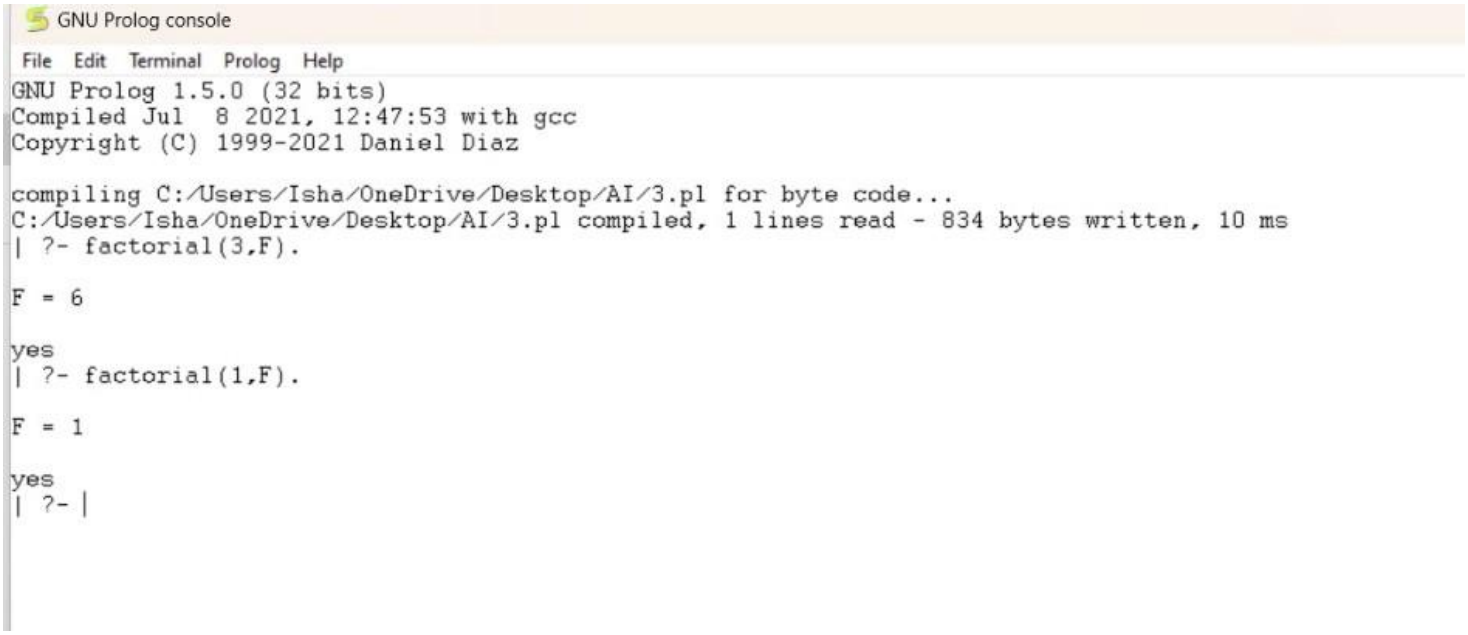
sum(A,B,S) :- S is A + B.

2. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers Xand Y.

max(X,Y,M) :- X>Y , M is X.
max(X,Y,M) :- X<Y , M is Y.
max(X,Y,M) :- X=Y , write('both are equal').

3. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of anumber N.

factorial(0,1).
factorial(N,F):-N1 is N-1 , factorial(N1,Y) , F is Y*N,!.



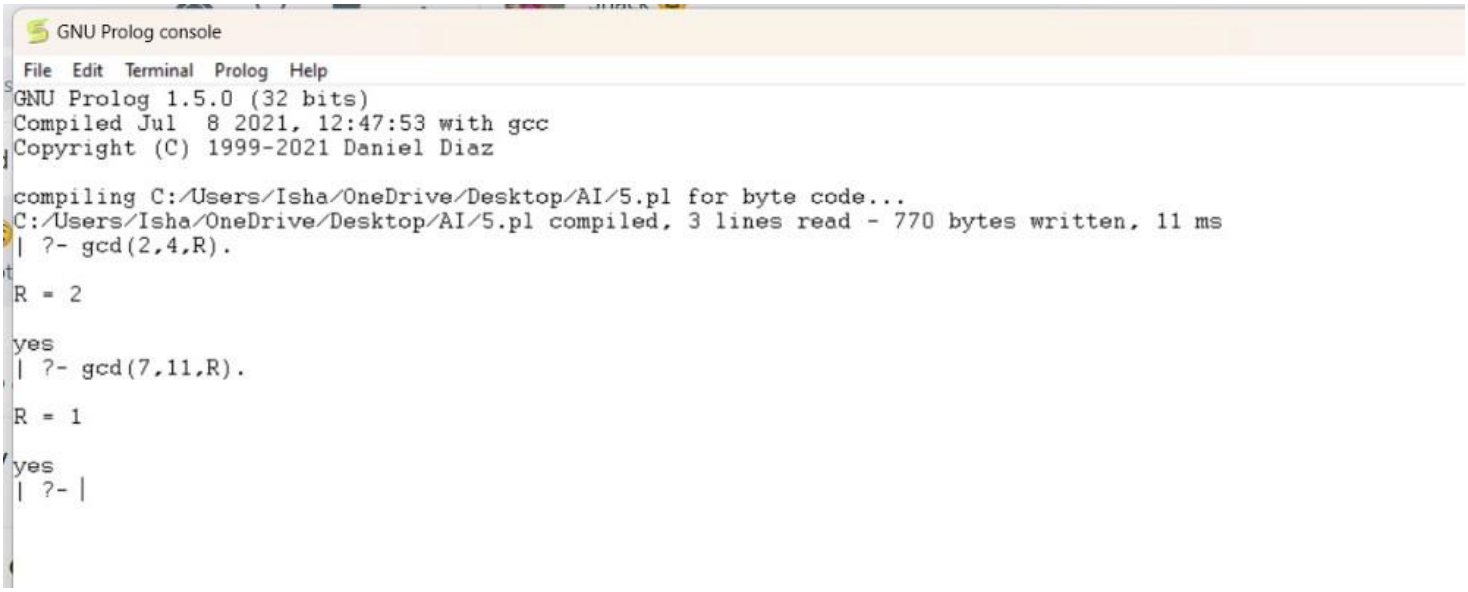
4. Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term ofthe fibonacci series.

generate_fib(1,0).
generate_fib(2,1).
generate_fib(N,T):- N1 is N-1 , N2 is N-2 , generate_fib(N1,T1) , generate_fib(N2,T2) , T is T1+T2,!.

5. Write a Prolog program to implement GCD of two

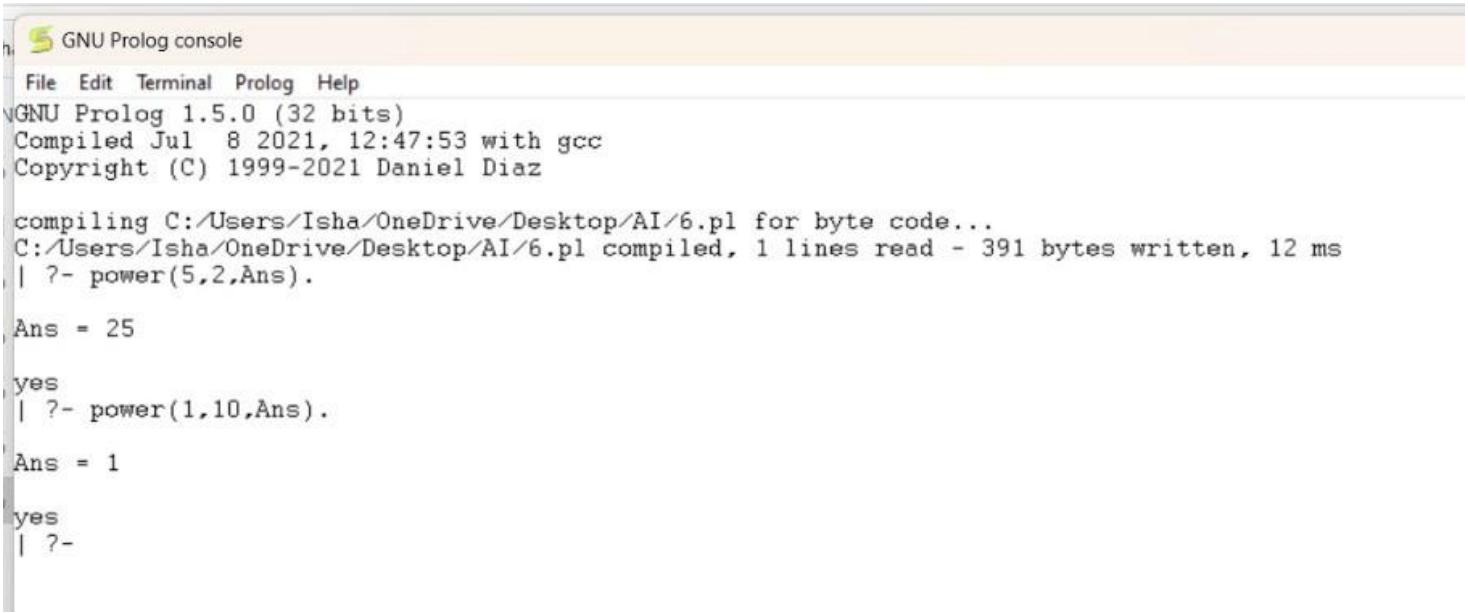
numbers.

```
gcd(0,A,A) :- !.
gcd(A,0,A) :- !.
gcd(A,B,R) :- B1 is mod(A,B) , gcd(B,B1,R).
```



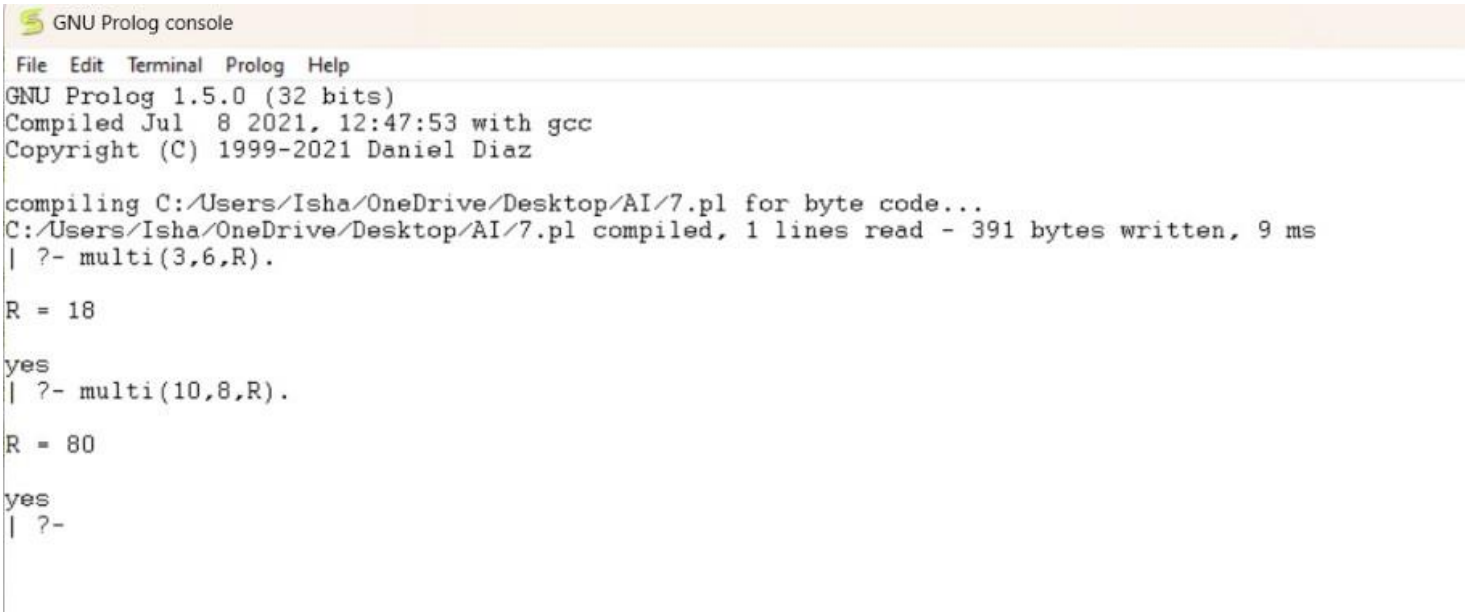
6. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to thepower Pow to get Ans.

```
power(Num,Pow,Ans) :- Ans is Num^Pow.
```



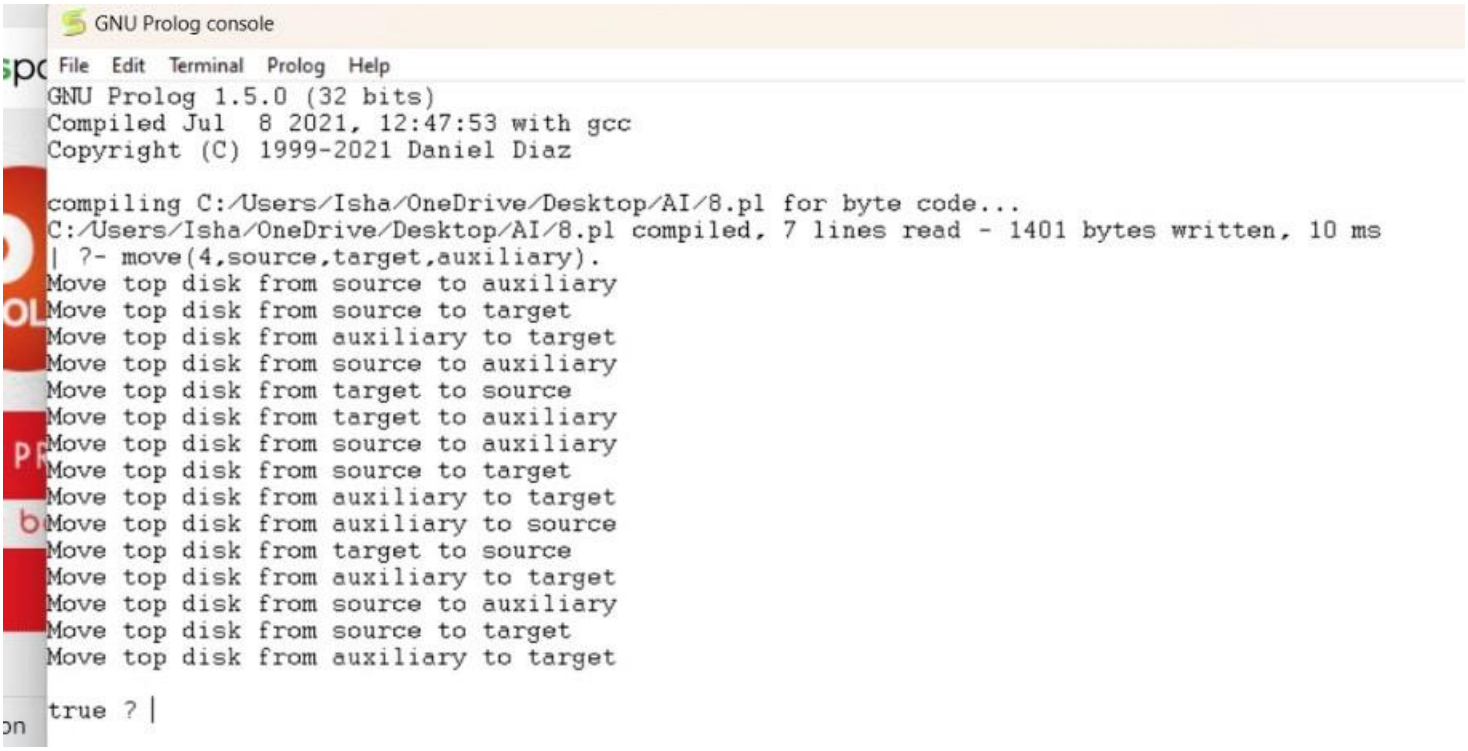
7. Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to bemultiplied and R represents the result.

multi(N1,N2,R) :- R is N1*N2.



8. Write a program in PROLOG to implement towerofhanoi (N) where N represents the number ofdiscs

```
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(1,X,Y,_),
   move(M,Z,Y,X).
```



9. Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] whereedge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a program to check whether there is a route from one node to another node.

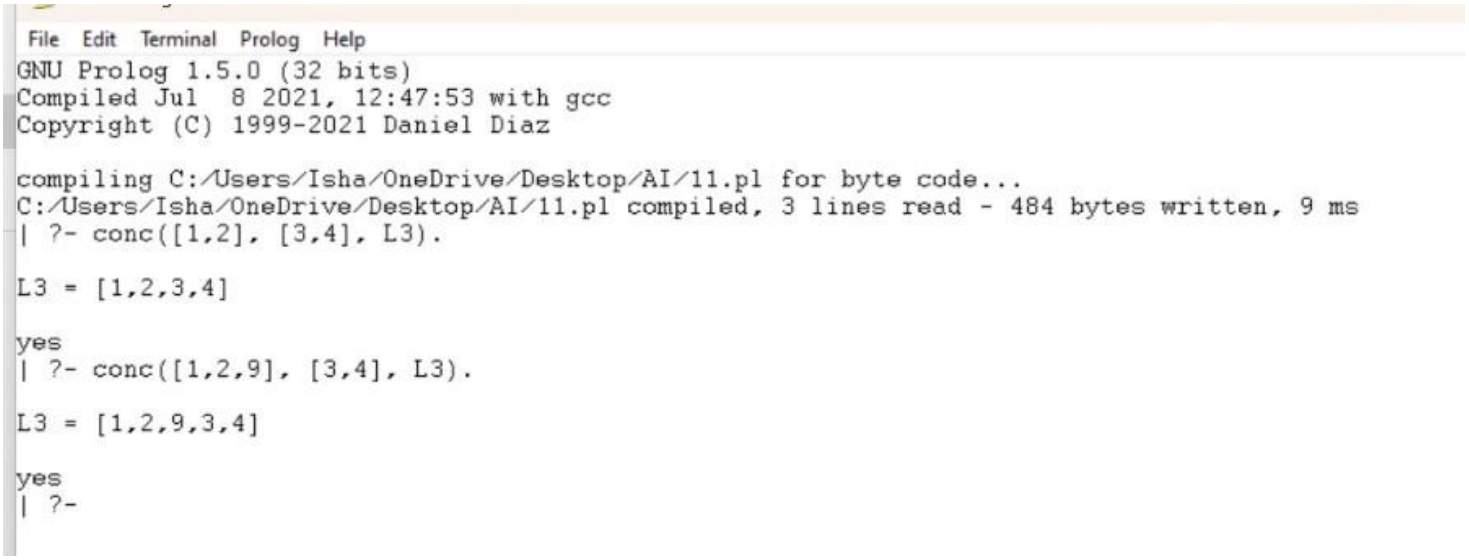
```
edge(p, q).
edge(q, r).
edge(q, s).
edge(s, t).
edge(q, r).

route(Node1, Node2) :-
    dfs(Node1, Node2, [Node1]).

dfs(Node, Node, _) :- !.
dfs(Node1, Node2, Visited) :-
    edge(Node1, X),
    not(member(X, Visited)),
    dfs(X, Node2, [X|Visited]).
```

10. **Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not**

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

11. Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended withL1 to get the resulted list L3.

```
conc([], L2, L2).
conc([H|T], L2, [H|L3]) :- conc(T, L2, L3).
```

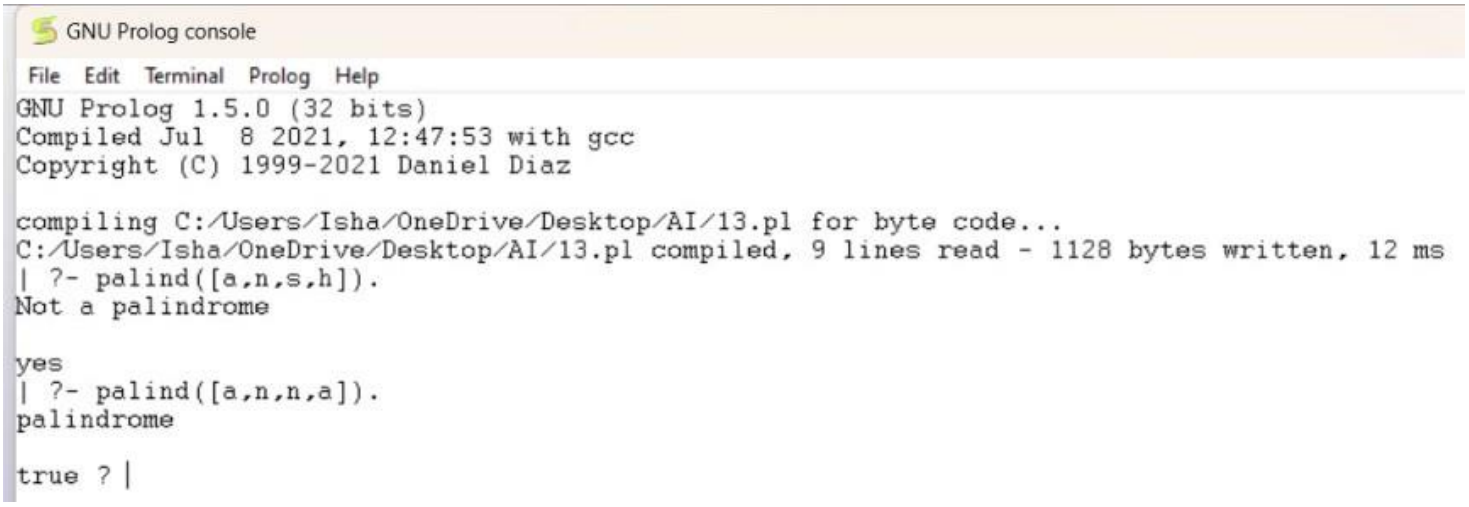


12. Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversedlist.

```
reverse([], []).
reverse([H|T], R) :- reverse(T, RevT), append(RevT, [H], R).
```

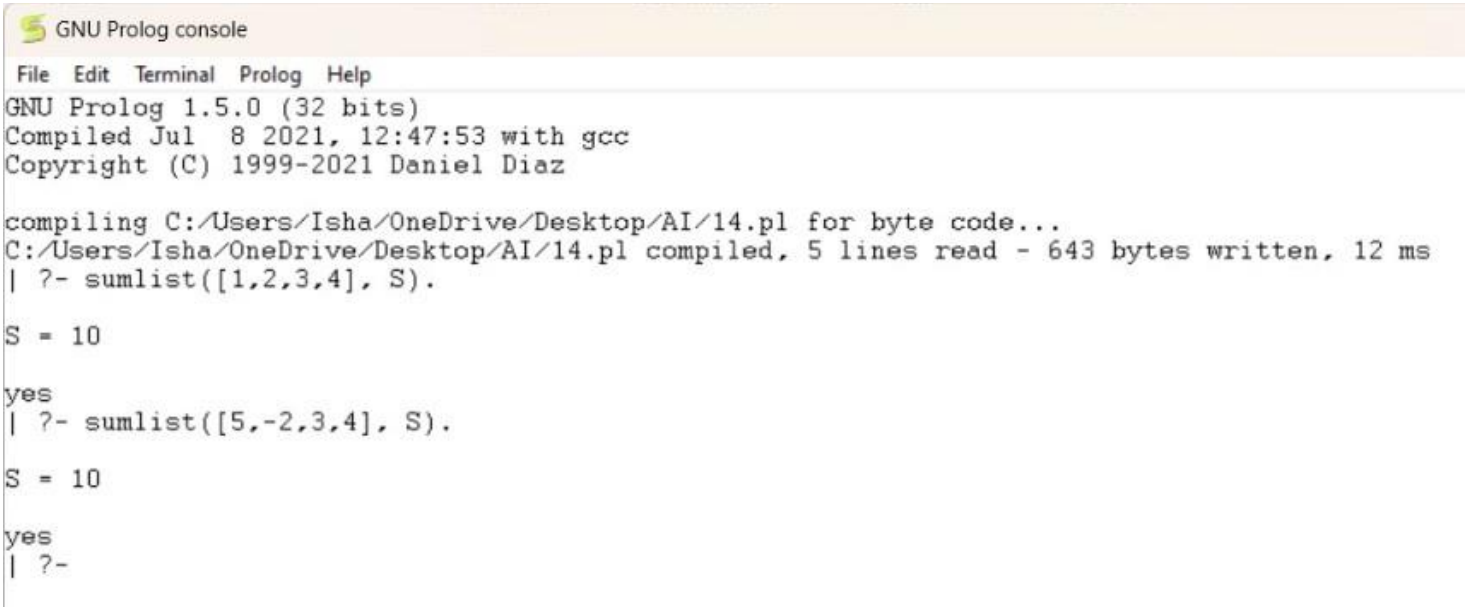
13. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is apalindrome or not.

```
palind([]):- write('palindrome').
palind([_]):- write('palindrome').
palind(L) :-
 append([H|T], [H], L),
 palind(T)
 ;
write('Not a palindrome').
```



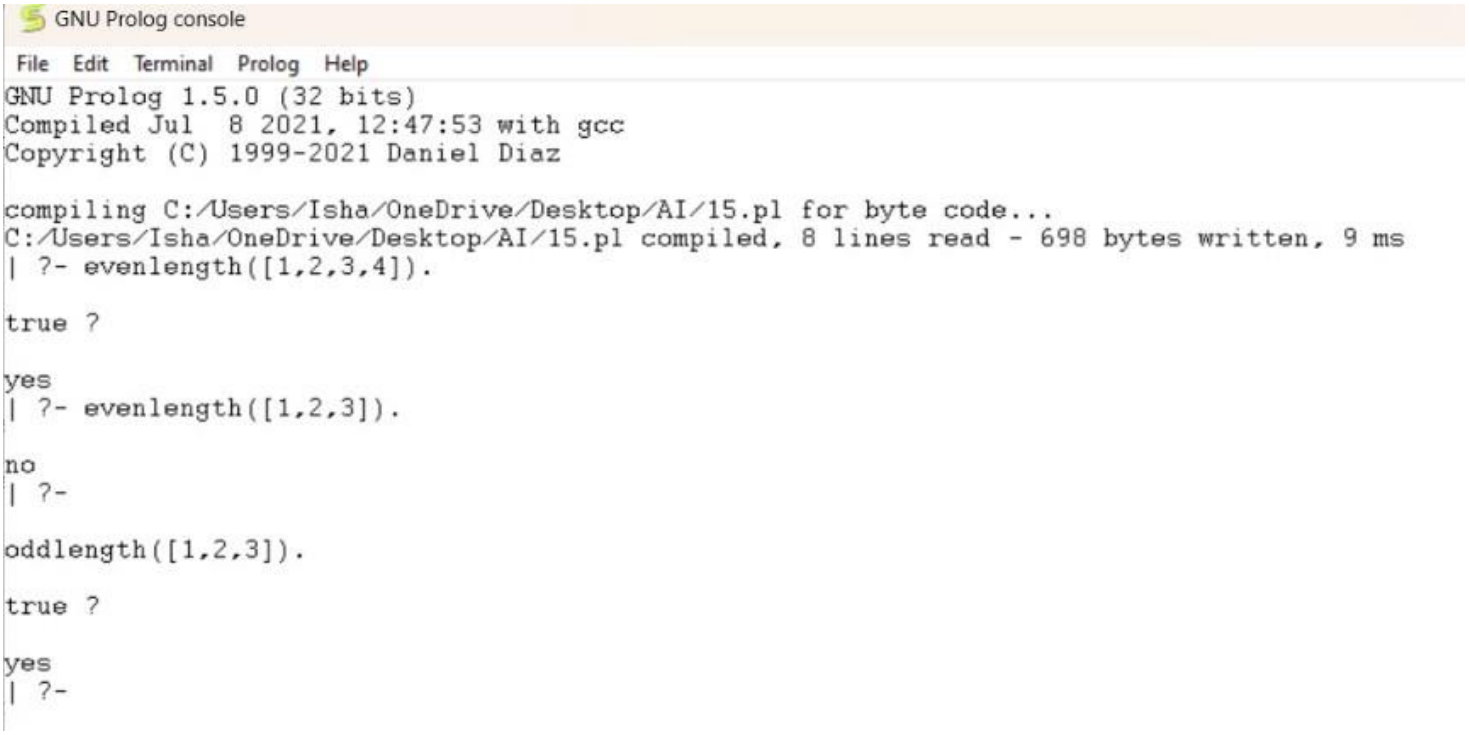
14. Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.

```
sumlist([], 0).

sumlist([H|T], S) :- sumlist(T, Rest), S is H + Rest.
```



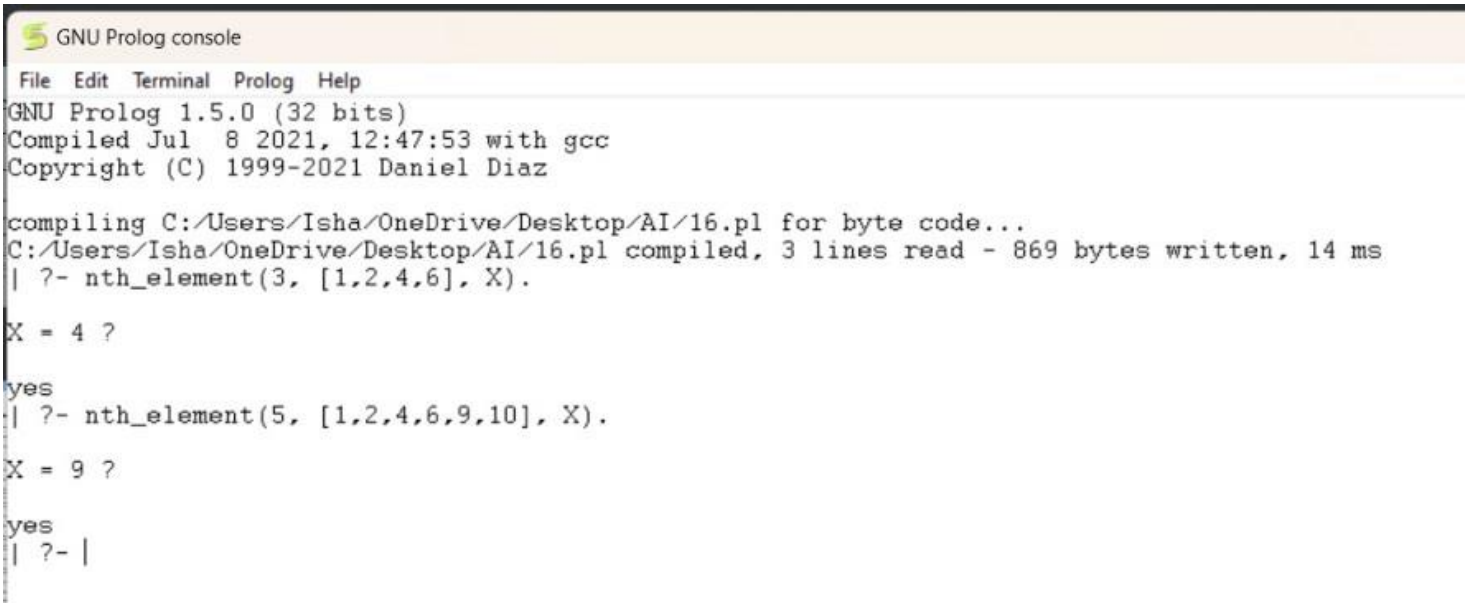
15. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) sothat they are true if their argument is a list of even or odd length respectively

```
evenlength([]).
evenlength([_|T]) :- oddlength(T).

oddlength([_]).
oddlength([_|T]) :- evenlength(T).
```



16. Write a Prolog program to implement nth_element (N, L, X) where N is the desired position, L is a list and X represents the Nth element of L.

```
nth_element(1, [X|_], X).
nth_element(N, [_|T],
X) :-N > 1,
N1 is N-1,
nth_element(N1, T, X).
```



17. Write a program in PROLOG to implement remove_dup (L, R) where L denotes the list with someduplicates and the list R denotes the list with duplicates removed.

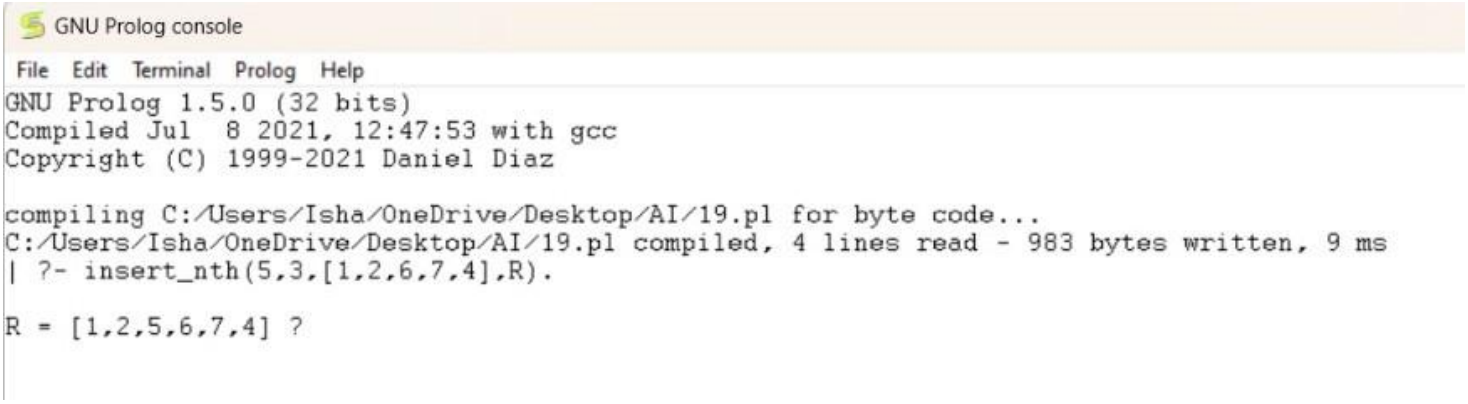
```
member(X,[X|_]).
member(X,[_|Y]):-
member(X,Y).
remove_dup(L,M):
-dupacc(L,[],M).
dupacc([],A,A).
dupacc([H|T],A,L):
-
member(H,A),dup
acc(T,A,L),!.
dupacc([H|T],A,L):
-
dupacc(T,[H|A],L).
```

18. **Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list.**

```
maxlist([X], X).
maxlist([H|T], M) :-
    maxlist(T,  M1),
    M is max(H, M1).
```

19. Write a prolog program to implement insert_nth(I, N, L, R) that inserts an item I into Nthposition of list L to generate a list R.

```
insert_nth(I, 1, L, [I|L]).
insert_nth(I, N, [H|T], [H|R]) :-
    N > 1,
    N1 is N-1,
    insert_nth(I, N1, T, R).
```



20. Write a Program in PROLOG to implement sublist(S, L) that checks whether the list S is thesublist of list L or not. (Check for sequence or the part in the same order).

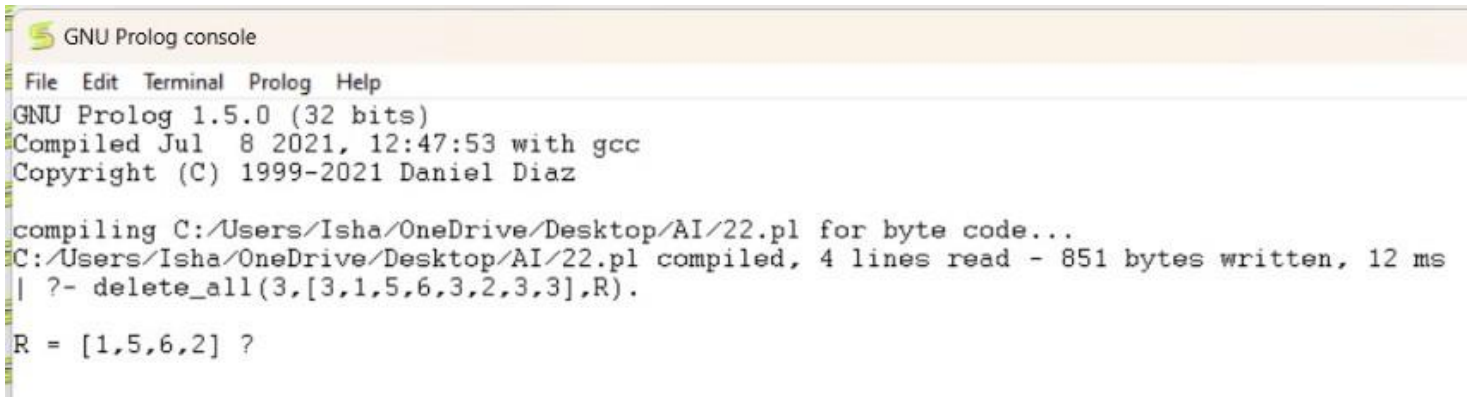
```
sublist([], _).
sublist([H|T], L) :-
    append(_, [H|L2], L),
    sublist_helper(T, L2).

sublist_helper([], _).
sublist_helper([H|T], [H|L2]) :-
    sublist_helper(T, L2).
```

21. Write a Prolog program to implement delete_nth (N, L, R) that removes the element on Nthposition from a list L to generate a list R.

```
delete_nth(1, [_|T], T).
delete_nth(N, [H|T], [H|R]) :-
    N > 1,
    N1 is N-1,
    delete_nth(N1, T, R).
```

22. Write a program in PROLOG to implement delete_all (X, L, R) where X denotes the elementwhose all occurrences has to be deleted from list L to obtain list R.

```
delete_all(_, [], []).
delete_all(X, [X|T], R) :-
    delete_all(X, T, R).
delete_all(X, [H|T], [H|R]) :-
    X \= H,
    delete_all(X, T, R).
```



23. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2is second ordered list and L3 represents the merged list.

```
merge([], L, L).
merge(L, [], L).
merge([H1|T1], [H2|T2], [H1|T]) :-
    H1 =< H2,
    merge(T1, [H2|T2], T).
merge([H1|T1], [H2|T2], [H2|T]) :-
    H1 > H2,
    merge([H1|T1], T2, T).
```

24. Write a PROLOG program that will take grammar rules in the following format:NT -> (NT | T)*
Where NT is any nonterminal, T is any terminal and Kleene star (*) signifies any number ofrepetitions, and generate the corresponding top-down parser, that is:
sentence -> noun-phrase, verb-phrasedeterminer -> [the]
will generate the following:

```
sentence (I, O) :- noun-phrase(I,R), verb-phrase (R,O).
determiner ([the|X], X) :- !.

sentence(I, O) :- noun_phrase(I, R), verb_phrase(R, O).

noun_phrase(I, O) :- article(I, R), noun(R, O).
noun_phrase(I, O) :- noun(I, O).

verb_phrase(I, O) :- verb(I, R), noun_phrase(R, O).
verb_phrase(I, O) :- verb(I, O).

article([the|X], X) :- !.

noun([dog|X], X) :- !.
noun([cat|X], X) :- !.

verb([chases|X], X) :- !.
verb([sees|X], X) :- !.
```

25. **Write a prolog program that implements Semantic Networks**

(ATN/RTN).

```
node(lion).
node(tiger).
node(animal).
node(feline).

rel(is_a, lion, animal).
rel(is_a, tiger, animal).
rel(is_a, feline, animal).
```

```
rel(is_a, lion, feline).
rel(is_a, tiger, feline).

?- rel(is_a, lion, animal).
Yes

?- rel(is_a, X, feline).
X = lion ;
X = tiger.
```

26. Write a prolog program that implements Semantic Networks (ATN/RTN).

```
node(lion).
node(tiger).
node(animal).
node(feline).

rel(is_a, lion, animal).
rel(is_a, tiger, animal).
rel(is_a, feline, animal).
rel(is_a, lion, feline).
rel(is_a, tiger, feline).

?- rel(is_a, lion, animal).
Yes

?- rel(is_a, X, feline).
X = lion ;
X = tiger.
```