PROLOG

Practical File

SUBMITTED BY
HARSHUL KUMAR
14HCS4158

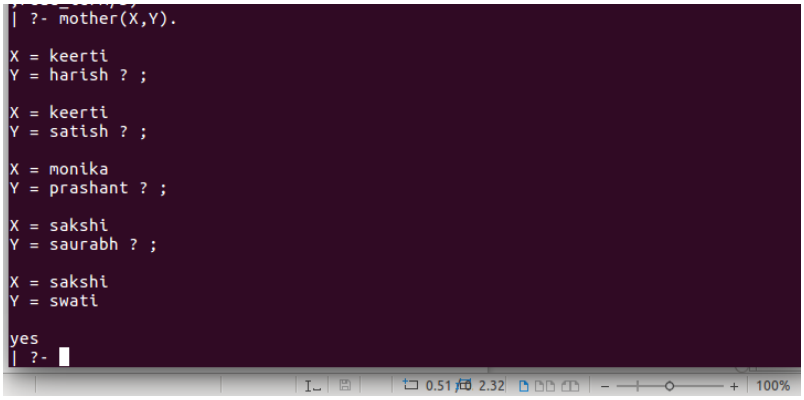
1. Write a prolog program to implement a family-tree.

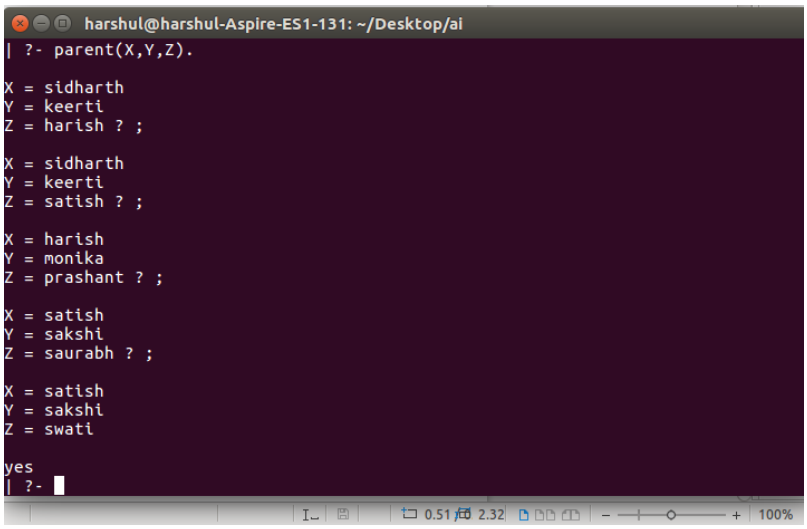
Code:-

```
male(sidharth).
        male(harish).
        male(satish).
        male(saurabh).
        male(prashant).

        female(keerti).
        female(monika).
        female(sakshi).
        female(swati).

        parent(sidharth,keerti,harish).
        parent(sidharth,keerti,satish).
        parent(harish,monika,prashant).
        parent(satish,sakshi,saurabh).
        parent(satish,sakshi,swati).

        brother(harish,satish).
        brother(satish,harish).
        brother(prashant,saurabh).
        brother(saurabh,prashant).
        sister(swati,saurabh).
        sister(swati,prashant).

    father(X,Y) :- parent(X,Z,Y).
    mother(X,Y) :- parent(Z,X,Y).
    son(X,Y,Z) :- male(X),father(Y,X),mother(Z,X).
    daughter(X,Y,Z) :- female(X),father(Y,X),mother(Z,X).
    wife(X,Y) :- female(X),parent(Y,X,Z).
    grandfather(X,Y) :- male(X),father(X,Z),father(Z,Y).
    uncle(X,Y) :- male(X),brother(X,Z),father(Z,Y).
    aunt(X,Y) :- wife(X,Z),uncle(Z,Y).
    cousin(X,Y) :- father(Z,X),brother(Z,W),father(W,Y).
    ancestor(X,Y,Z) :- parent(X,Y,Z).
    ancestor(X,Y,Z) :- parent(X,Y,W),ancestor(W,U,Z).
```

Output :


ms
yes
| ?- go.
uncaught exception: error(existence_error(procedure,go/0),top_level/0)
| ?- father(X,Y).
X = sidharth
Y = harish ? ;
X = sidharth
Y = satish ? ;
X = harish
Y = prashant ? ;
X = satish
Y = saurabh ? ;
X = satish
Y = swati
yes
| ?-


| ?- mother(X,Y).
X = keerti
Y = harish ? ;
X = keerti
Y = satish ? ;
X = monika
Y = prashant ? ;
X = sakshi
Y = saurabh ? ;
X = sakshi
Y = swati
yes
| ?-


harshul@harshul-Aspire-ES1-131: ~/Desktop/ai
| ?- parent(X,Y,Z).
X = sidharth
Y = keerti
Z = harish ? ;
X = sidharth
Y = keerti
Z = satish ? ;
X = harish
Y = monika
Z = prashant ? ;
X = satish
Y = sakshi
Z = saurabh ? ;
X = satish
Y = sakshi
Z = swati
yes
| ?-

```
| ?- son(prashant,X,Y).

X = harish
Y = monika ? ;

no
| ?- daughter(swati,X,Y).

X = satish
Y = sakshi

yes
| ?- grandfather(X,Y).

X = sidharth
Y = prashant ? ;

X = sidharth
Y = saurabh ? ;

X = sidharth
Y = swati ? ;

no
| ?-
```

```
harshul@harshul-Aspire-ES1-131: ~/Desktop/ai
Y = prashant ? ;

X = sidharth
Y = saurabh ? ;

X = sidharth
Y = swati ? ;

no
| ?- aunty(X,Y).
uncaught exception: error(existence_error(procedure,aunty/2),top_level/0)
| ?- aunt(X,Y).

X = monika
Y = saurabh ? ;

X = monika
Y = swati ? ;

X = sakshi
Y = prashant ? ;

X = sakshi
Y = prashant ? ;

no
| ?-
```

```
harshul@harshul-Aspire-ES1-131: ~/Desktop/ai
| ?- aunty(X,Y).
uncaught exception: error(existence_error(procedure,aunty/2),top_level/0)
| ?- aunt(X,Y).

X = monika
Y = saurabh ? ;

X = monika
Y = swati ? ;

X = sakshi
Y = prashant ? ;

X = sakshi
Y = prashant ? ;

no
| ?- ancestor(X,Y,prashant).

X = harish
Y = monika ? ;

X = sidharth
Y = keerti ? ;

no
| ?-
```

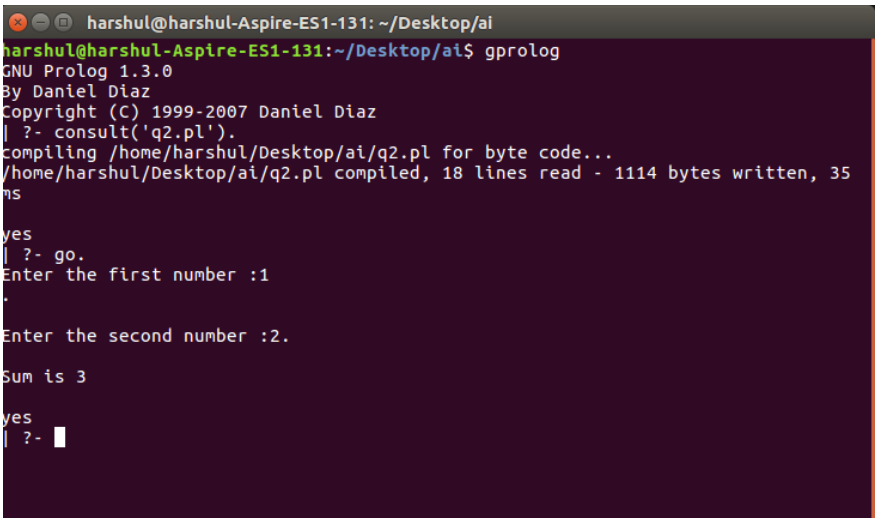
2. Write a prolog program to calculate the sum of two numbers.

Code:-

go:-

```
write('Enter the first number :'),read(X),nl,
write('Enter the second number :'),read(Y),nl,

sum(X,Y,Re),

write('Sum is '),write(Re).

sum(X,Y,Re):- Re is X+Y.


/*
    sum(X,Y,Re) is a predicate.
    X and Y are input, Re is the result.
*/
```

Output:-

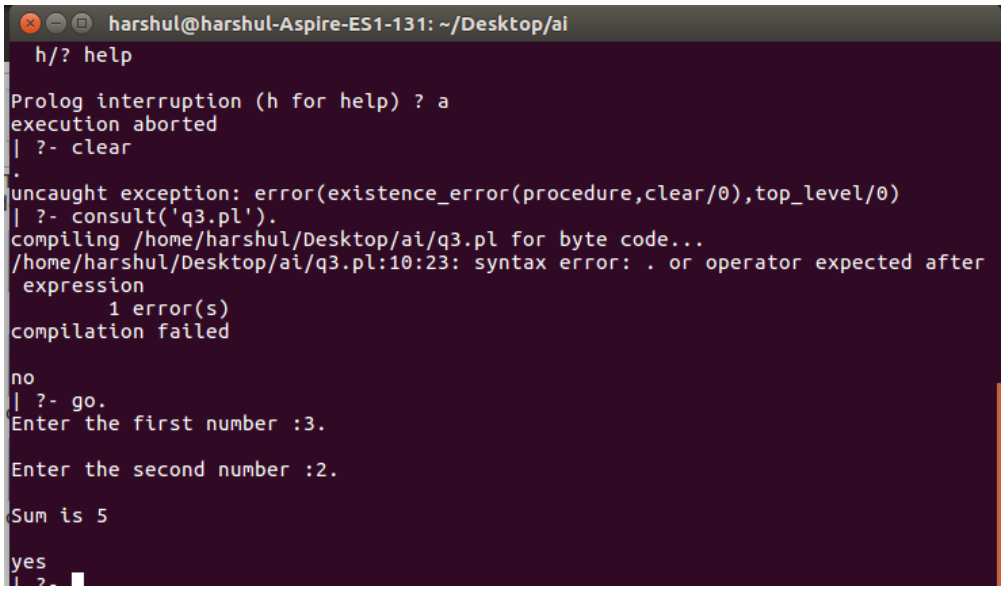


3. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X
and Y.

Code:-

```
go:-
        write('Enter the first number :'),read(X),nl,
        write('Enter the second number :'),read(Y),nl,

        sum(X,Y,Re),

        write('Maximum number is '),write(Re).


max(X,Y,Re):- Re is X =< Y, Re is X => Y.
max(X,0,Re):- Re is X.
max(0,Y,Re):- Re is Y.


/*

        sum(X,Y,Re) is a predicate.
        X and Y are input, Re is the result.

*/
```

Output:-

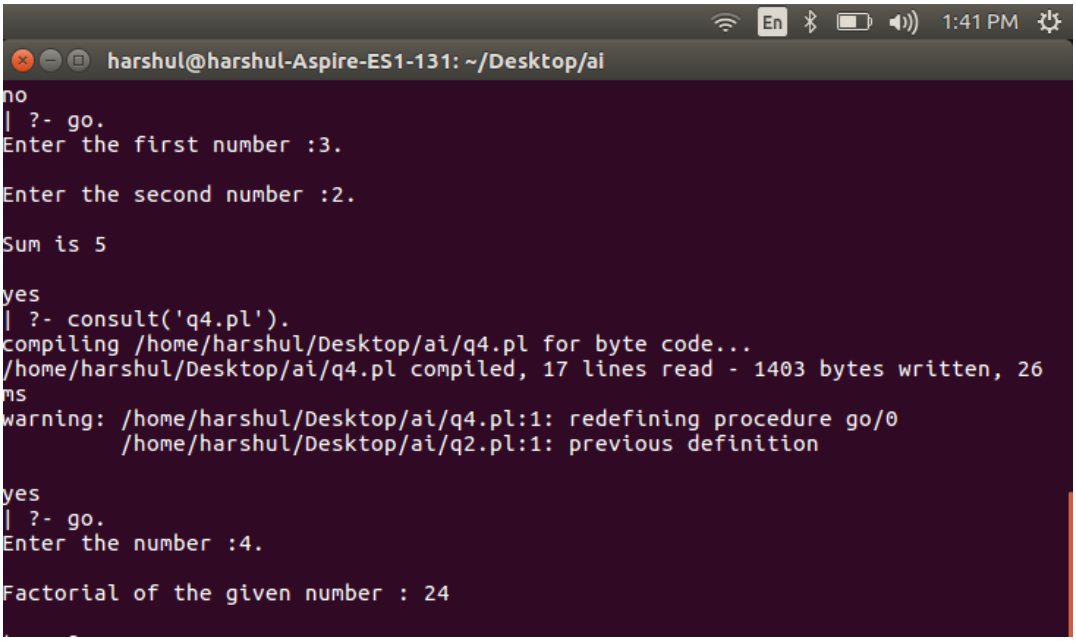


4. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.

Code:-

```
go:-
        write('Enter the number :'),read(X),nl,
        fact(X,Re),
        write('Factorial of the given number : '),write(Re),nl.


        fact(0,1).
        fact(1,1).
        fact(N,R):- N1 is N-1, fact(N1,Factorial), R is Factorial*N.


        /*
        fact(X,R). is the predicate used for calculation of factorial of 'X'.
        X is the input
        */
```

Output:-

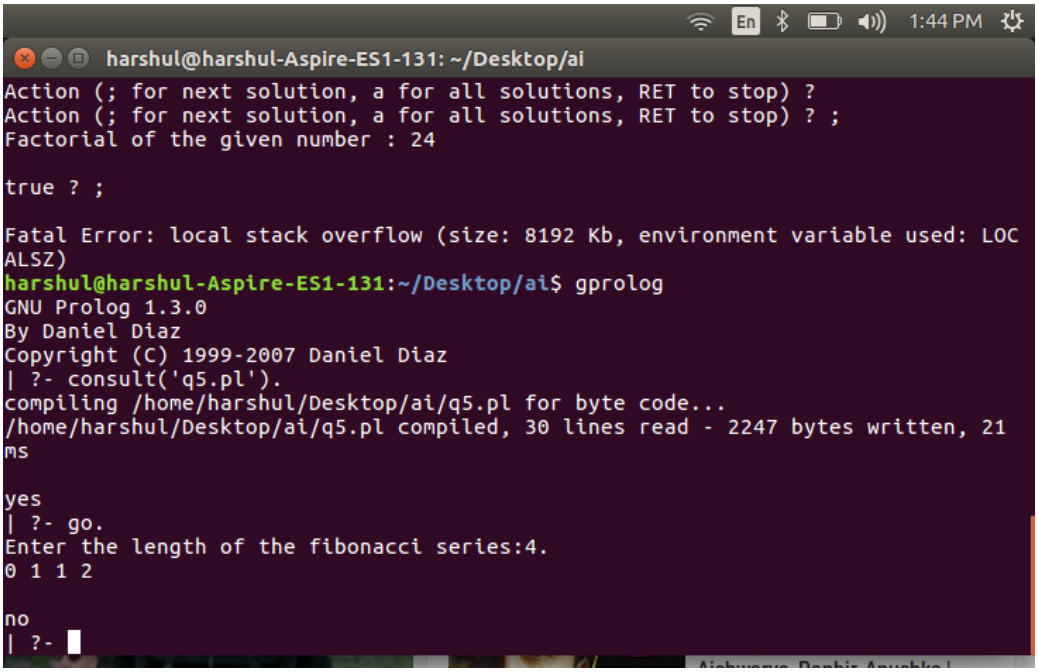


5. Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the fibonacci series.

Code:-

```
go:-
        write('Enter the length of the fibonacci series:'),
        read(X),
        for(1,X).

        for(I,N):-      I=<N,
                        gen_fib(I,F),
                        write(F),
                        write(' '),
                        I1 is I+1,
                        for(I1,N).

        gen_fib(1,0):-!.
        gen_fib(2,1):-!.
        gen_fib(N,F):-                  N>2,
                                N1 is N-1,
                                N2 is N-2,
                                gen_fib(N1,F1),
                                gen_fib(N2,F2),
                                F is F1+F2.
        /*
        gen_fib(N,F). is the predicate which calculates ' (N-1)+(N-2) ' and stores its result in 'F'.
        'X' is the input or the length of fibonacci series to be printed.
        ' I=<N ' is the terminating condition for 'For Loop'.
        */
```

Output:-

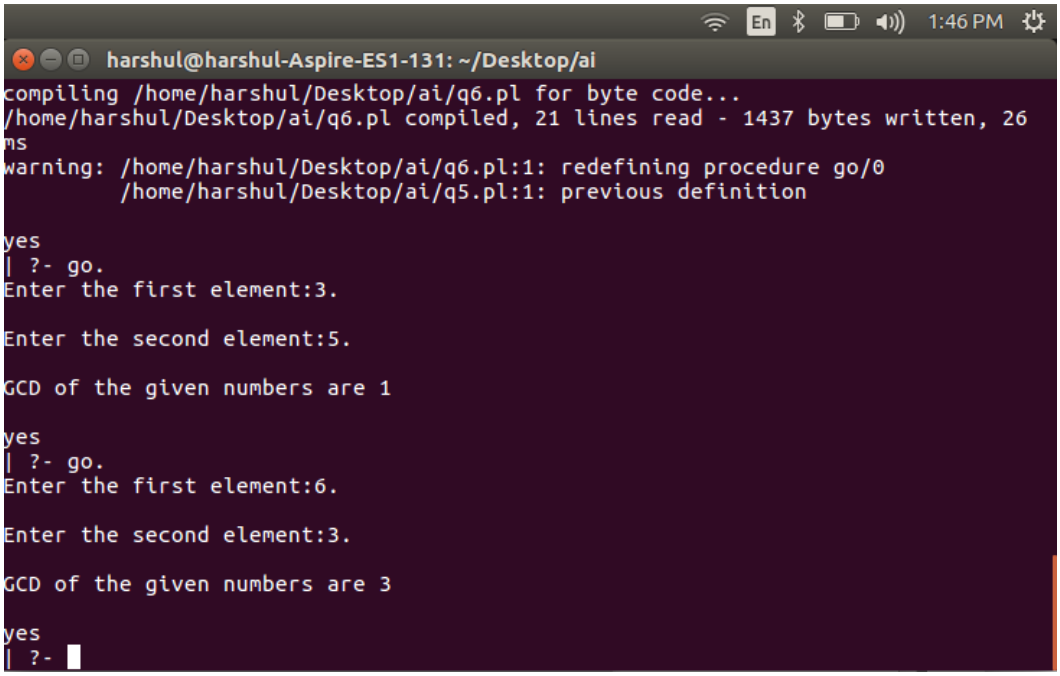


6. Write a Prolog program to implement GCD of two numbers.

Code:-

```
go:-
        write('Enter the first element:'),
        read(A),nl,
        write('Enter the second element:'),
        read(B),nl,
        write('GCD of the given numbers are '),
        gcd(A,B,C),write(C).


        gcd(X,0,X):- !.
        gcd(X,Y,Z):-  N is X mod Y,
                      gcd(Y,N,Z).



/*

        gcd(X,Y,Re) is a predicate used to calculate GCD.
        X and Y are input, Z is the result.

*/
```

Output:-



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

Code:-

```
go:-
        write('Enter the list :'),read(Y),nl,
        write('Enter the number to be searched :'),read(X),nl,

        member(X,Y).

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

/*

        member(X,Y). is the predicate .
        '[ X | T ]' in this 'X' is the head ( single element ) of the list
        and 'T' is the rest part of the list ( single or rest element).

        Y is the list and X is the number going to be serach in the list.

*/
```

Ouput:-

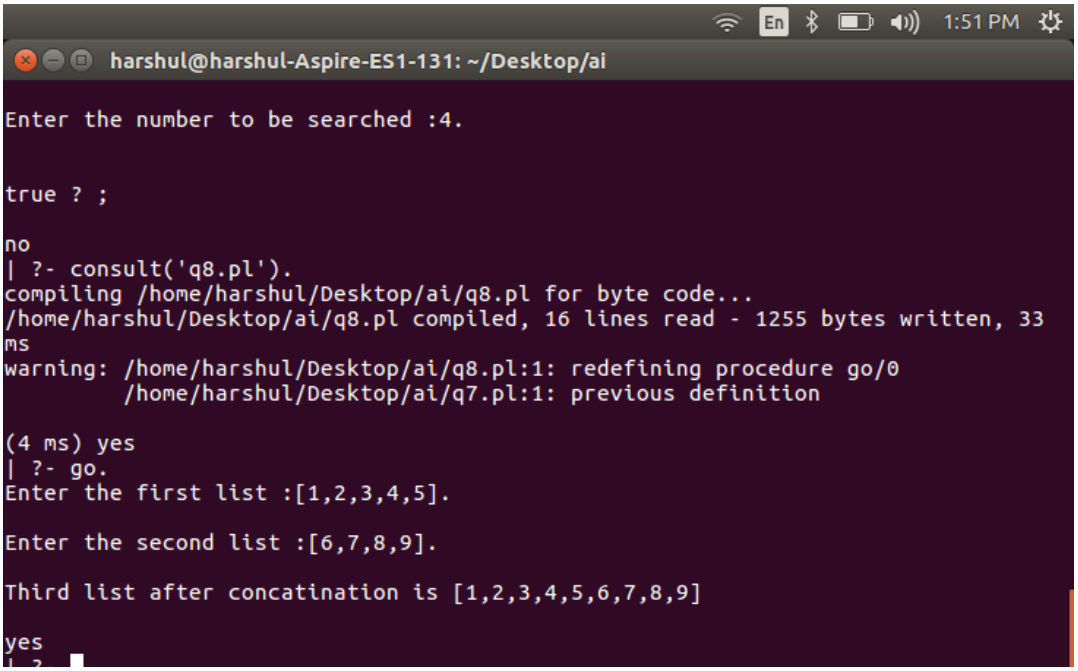
8. Write a Prolog program to implement conc (L1, L2, L3) where L2 is the list to be appended with

L1 to get the resulted list L3.

Code:-

```
go:-
        write('Enter the first list :'),read(X),nl,
        write('Enter the second list :'),read(Y),nl,
        conc(X,Y,Re),
        write('Third list after concatination is '),write(Re).

conc([],List,List).
conc([X|L1],L2,[X|L3]):- conc(L1,L2,L3).


/*

        conc(X,Y,Re). is the predicate used for concatination.
        'X' and 'Y' are the two distinct list and 'Re' is the concatinated result of the two inputs.

*/
```

Output:-



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

reversed list.

Code:-

```
go:-
        write('Enter the list :'),read(X),nl,
        reverse(X,Y),
        write(Y),nl.



reverse([],[]).  %reverse of empty is empty - base case
reverse([H|T], RevList):-
reverse(T, RevT), conc(RevT, [H], RevList).  %concatenation

/*
        reverse(X,Y). is the predicate used.
        'X' is the inputed list and 'Y' is the reverse of the same list.

*/
```

Output:-

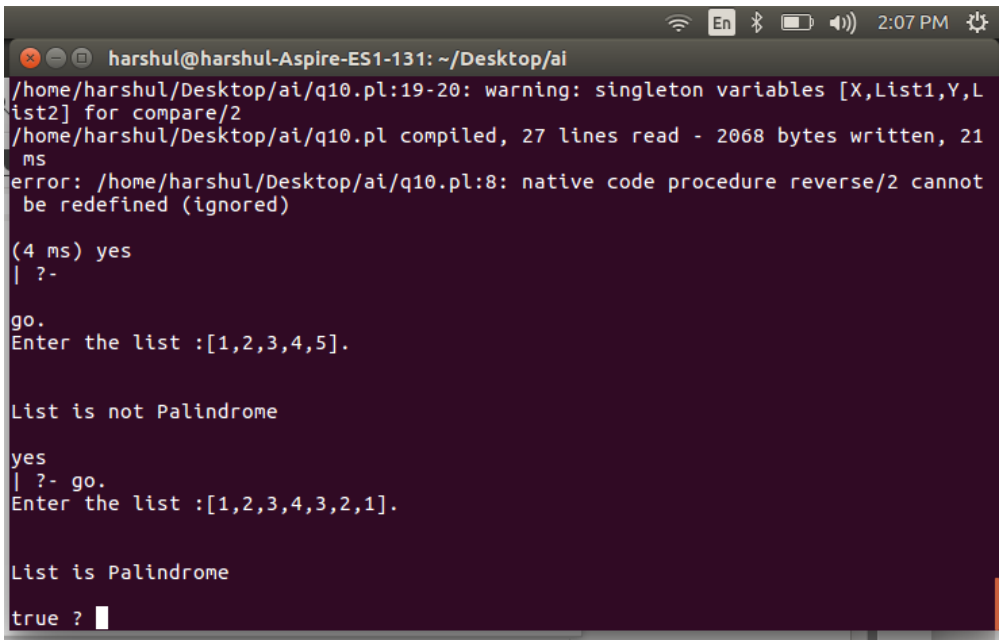
10. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a

palindrome or not.

Code:-

```
go:-
        write('Enter the list :'),read(X),nl,
        palin(X),nl.

   palin(List1):-
      reverse(List1,List2),
      compare(List1,List2).
         reverse([],[]).  %reverse of empty is empty - base case
         reverse([H|T], RevList):-
         reverse(T, RevT), conc(RevT, [H], RevList).  %concatenation


   compare([],[]):-
      nl,write('List is Palindrome'). %base condition

   compare([X|List1],[X|List2]):-
      compare(List1,List2).

   compare([X|List1],[Y|List2]):-
      nl,write('List is not Palindrome').        %base condition negation

/*
        plain(X), reverse(List1,List2) and compare(List1,List2) are the predicates used.
        X is the inputed list.
        List2 is the reverse of X.
        Compare will compare X and List2 and return wheather the list X is palindrom or not.
*/
```

Output:-

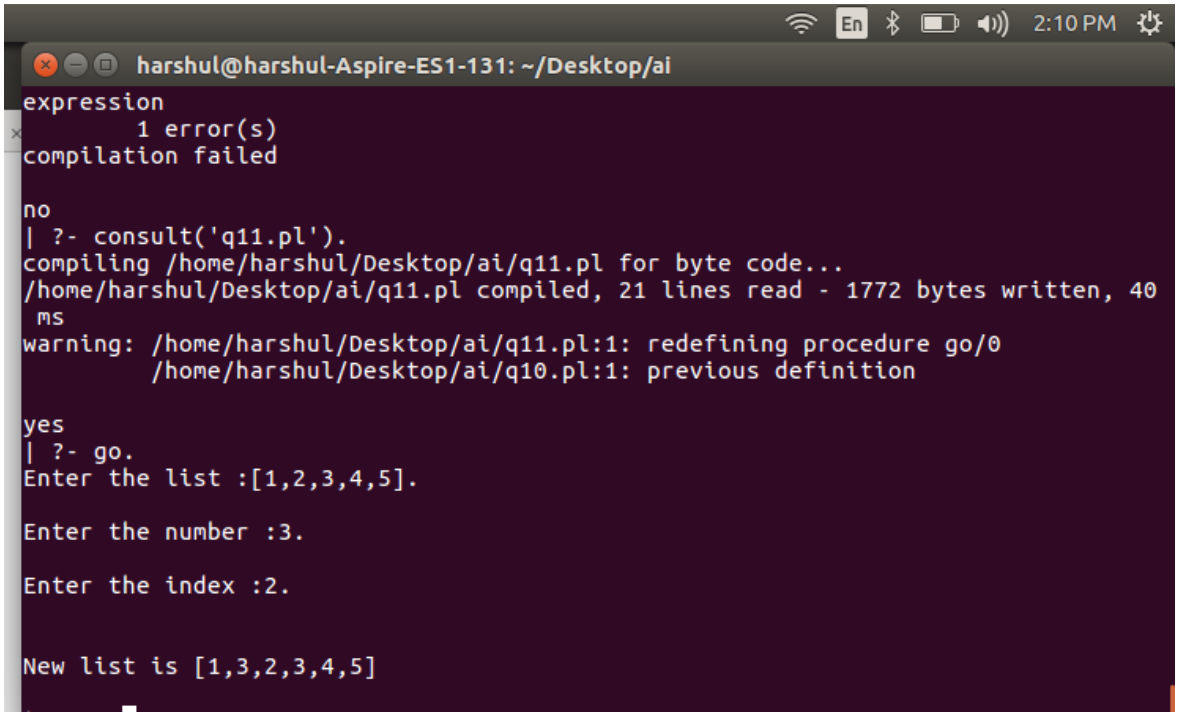


11. Write a prolog program to implement insert_nth(I, N, L, R) that inserts an item I into Nth position of list L to generate a list R.

Code:-

```
go:-
        write('Enter the list :'),read(L),nl
        write('Enter the number :'),read(V),nl,
        write('Enter the index :'),read(N),nl,
        ins(N,V,L,NL),nl,
        write('New list is '),write(NL).




ins(1,V,L,[V|L]).       %base condition
ins(N,V,[H|T],[H|T2]):- M is N-1, ins(M,V,T,T2).

/*
        ins(N,V,L,NL). is the predicate used.

        N is the index where new value is going insert.
        V is the value that is going to be inserted.
        L is the list that is going to affected.
        NL is the new list with inserted value.
*/
```

Output:-

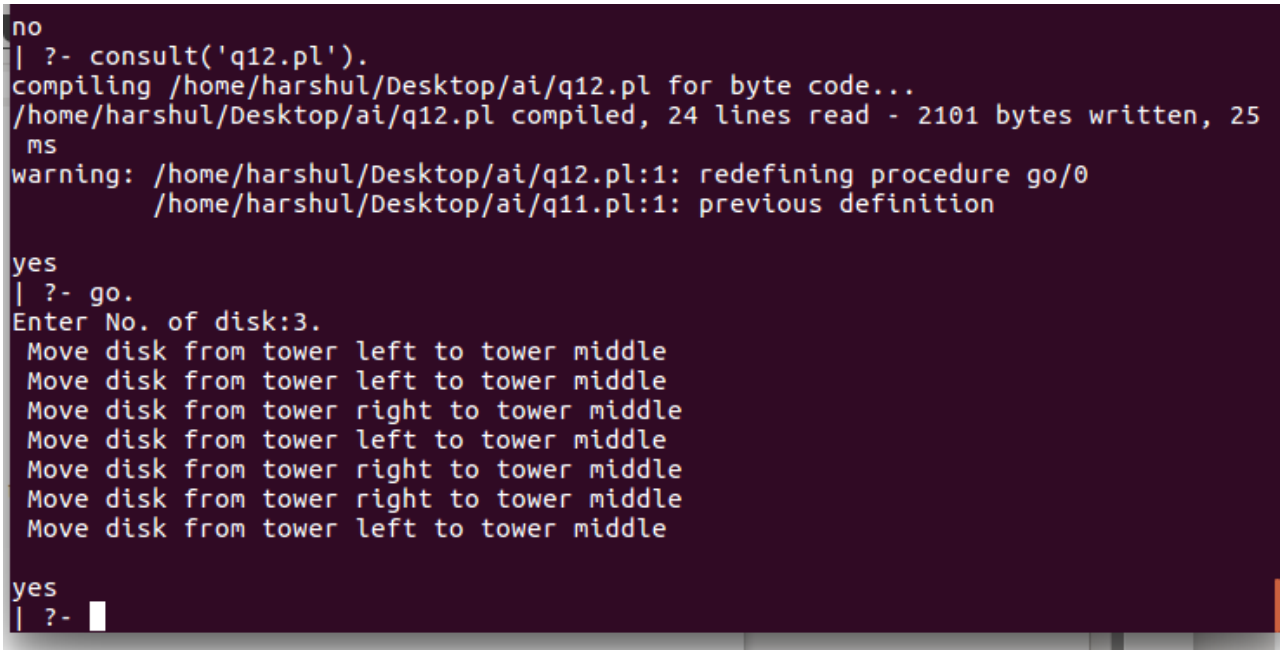


12. Write a program in PROLOG to implement towerofhanoi (N) where N represents the number of discs.

Output:-

```
go:-    write('Enter No. of disk:'),
        read(N),
        toh(N).

        toh(N):- mov(N,left,right,middle).

        mov(0,_,_,_):-!.
        mov(1,A,_,C):-inform(A,C),!.
        mov(N,A,B,C):- M is N-1,
                       mov(M,A,B,C),
                       inform(A,C),
                       mov(M,B,A,C).

        inform(A,B):- write(' Move disk from tower '),
                      write(A),
                      write(' to tower '),
                      write(B),nl.

/*
        toh(N), move(N,A,B,C), and inform(A,B) are the predicate used.
        N is the number of disks.


*/
```

Output:-



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

Code:-

```
go:-    write('Enter  the list consisting duplicates').
        nl,
        read(X),
        write('List after removing duplicates : '),
        nl,
        remove_duplicates(X,K),
        write(K).



remove_duplicates([],[]).
remove_duplicates([H|T],X) :-       member1(H,T),!,
                                    remove_duplicates(T,X).
remove_duplicates([H|T],[H|X]) :-   remove_duplicates(T,X).


member1(X,[H|_]) :- X==H,!.         %condition to check whether element is part of the list
member1(X,[_|T]) :- member1(X,T).


/*

        remove_duplicates(X,k) and member1(H,T) are the predicate used.
        X is the list with duplicate elements.
        K is the list without duplicates elements.

*/
```

Output:-

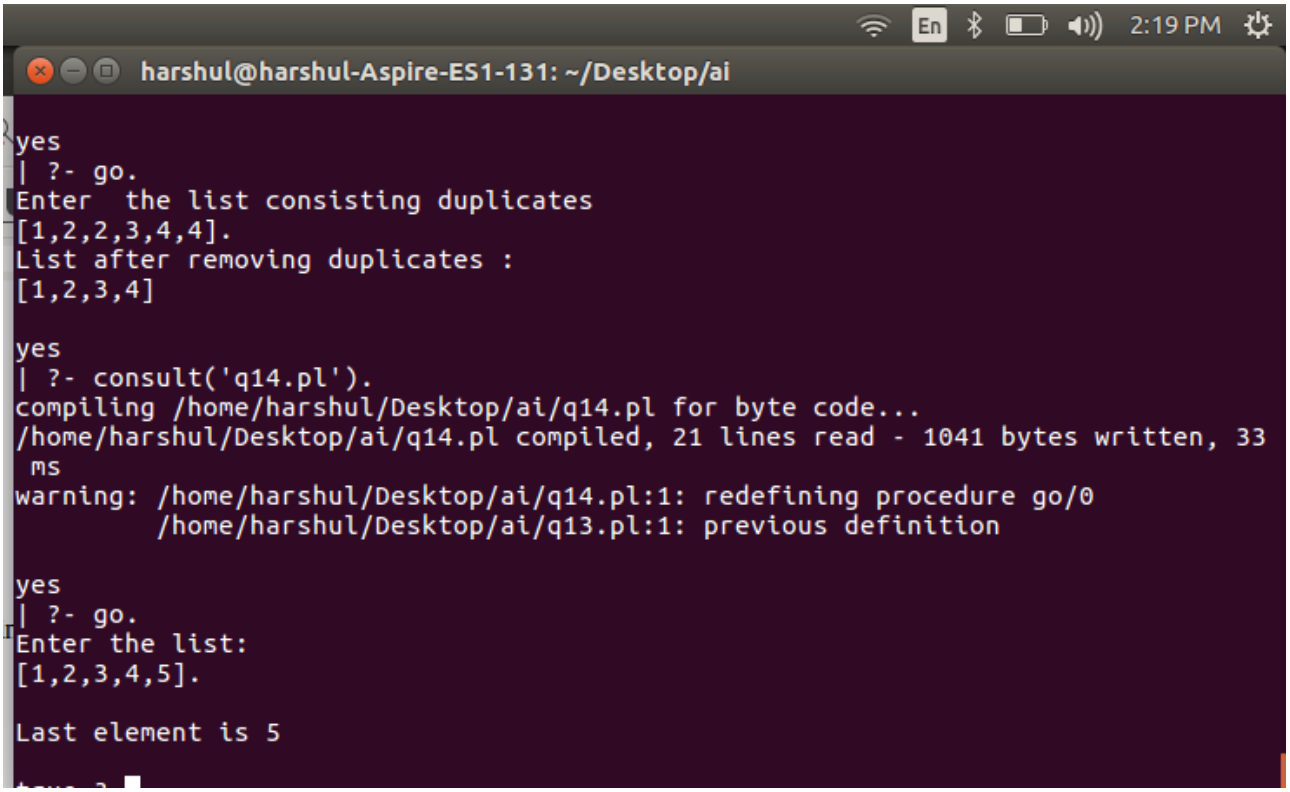
```
yes
| ?- consult('q13.pl').
compiling /home/harshul/Desktop/ai/q13.pl for byte code...
/home/harshul/Desktop/ai/q13.pl:1-7: warning: suspicious predicate (',')/2
/home/harshul/Desktop/ai/q13.pl compiled, 13 lines read - 2231 bytes written, 32
 ms

yes
| ?- go.
Enter  the list consisting duplicates
[1,2,2,3,4,4].
List after removing duplicates :
[1,2,3,4]

yes
| ?-
```

14. Write a Prolog program to implement last_el (L, X) where L is a list and X represents the last element of list L.

Code:-

```
go:-
        write('Enter the list:'),nl,
        read(L),nl,
        write('Last element is '),
        lastd(L,LE),
        write(LE).

         lastd([Head],X):-
      X is Head.                        %updating the lastelement
      lastd([_|Tail],X):-
        lastd(Tail,X).

/*
        lastd(L,LE). is the predicate used.
        L is the list .
        LE is the last element that we compute.


*/
```

Output:-

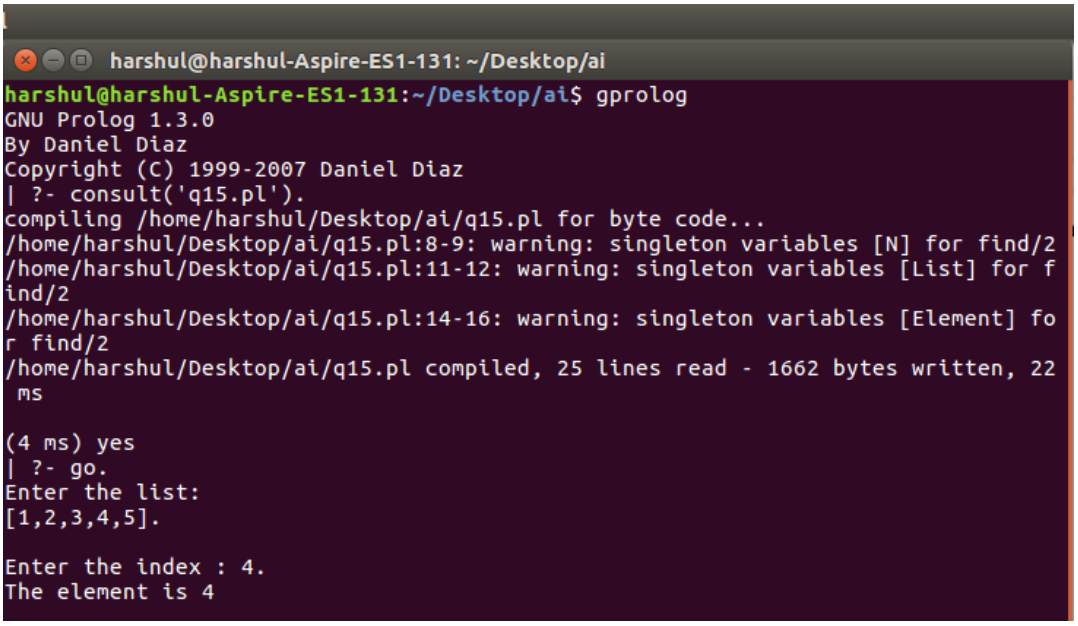


15. Write a Prolog program to implement nth_element (N, L, X) where N is the desired position, L is a list and X represents the Nth element of L.

Code:-

```
go:-
        write('Enter the list:'),nl,
        read(L),nl,
        write('Enter the index : '),
        read(ID),
        find(L,ID).

        find([],N) :-
    write('There is no such element in the list'),nl.

  find([Element|List],1) :-
    write('The element is '),write(Element),nl.

  find([Element|List],N) :-
    N1 is N-1,
    find(List,N1).


/*
        find(L,ID).is the predicate used
        L is the list.
        ID is the index of the element that we are fetching.


*/
```

Output:-

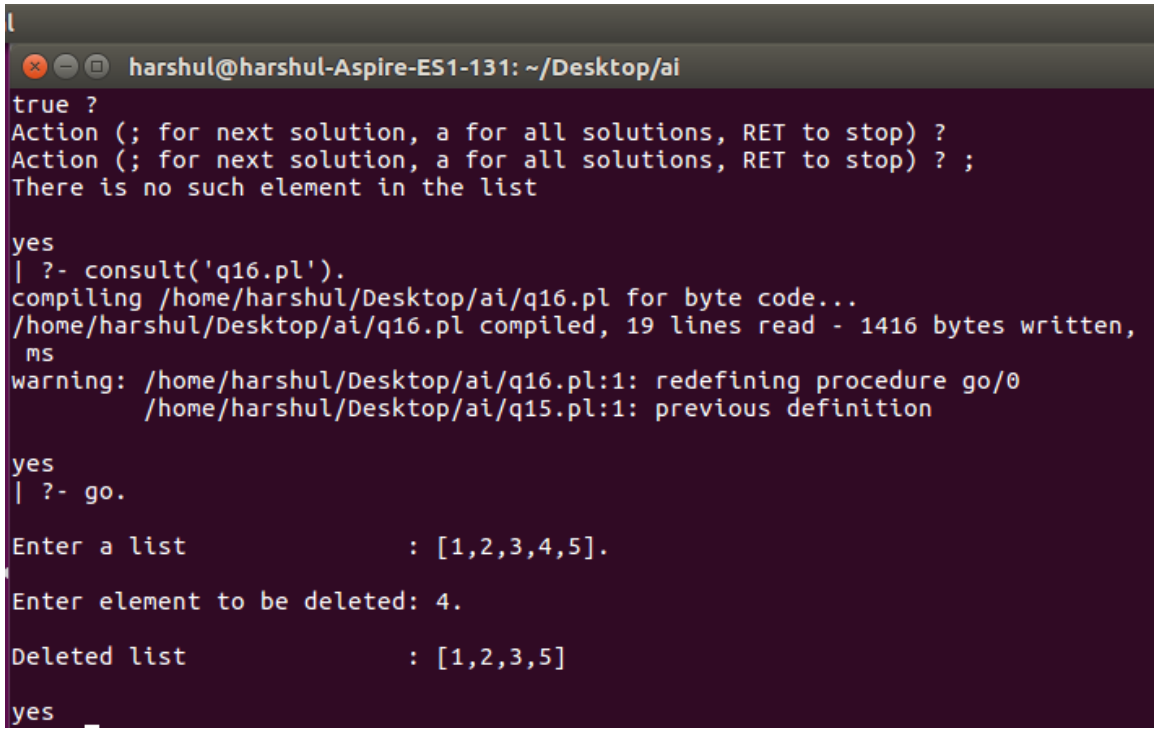


16. Write a Prolog program to implement delete_first (X, L ,R) where X denotes the element whose first occurrence has to be deleted from list L to obtain list R.

Code:-

```
go:-
 nl,write('Enter a list              : '),
 read(L),
 nl,write('Enter element to be deleted: '),
 read(E),
 delete_first(E,L,X),
 nl,write('Deleted list              : '),
 write(X).

delete_first(X,[X|T],T):-!.                    %element deleted
delete_first(X,[H|T],[H|T1]) :- delete_first(X,T,T1).

/*
        delete_first(E,L,X). is the predicate used.
        L is the list.
        E is the element whose first occurence is going to be deleted.
        X is the list after deletion.

 */
```

Output:-

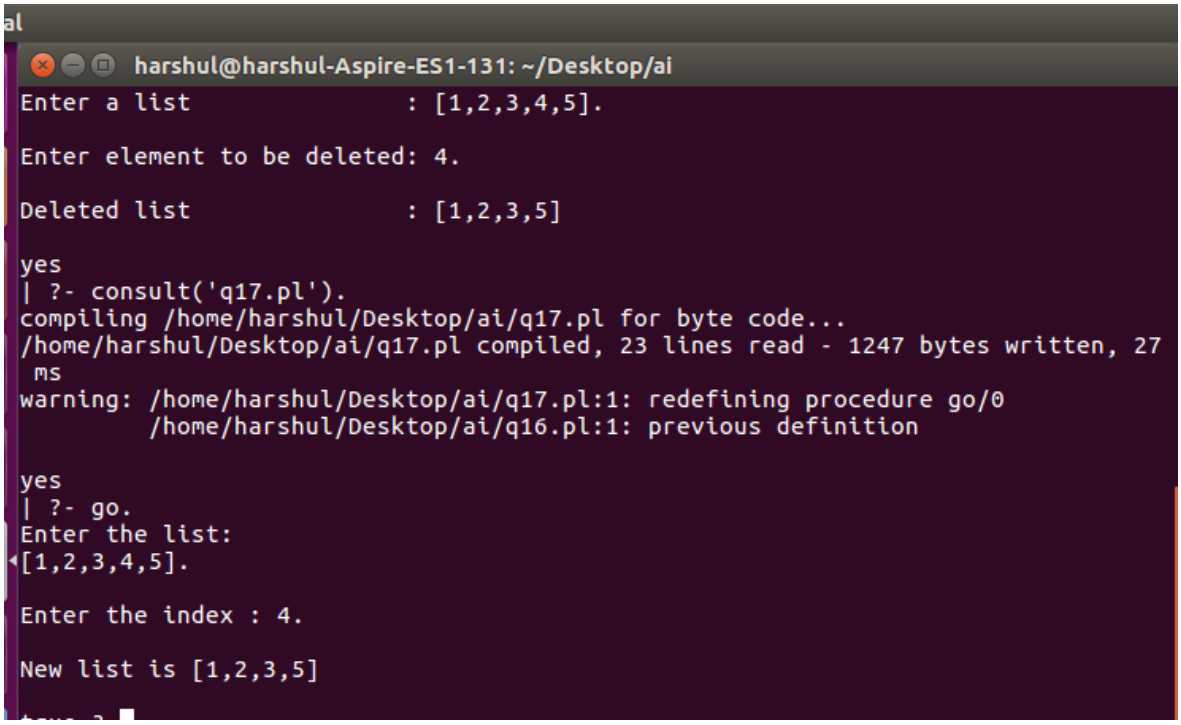


17. Write a Prolog program to implement delete_nth (N, L, R) that removes the element on Nth position from a list L to generate a list R.

Code:-

```
go:-
        write('Enter the list:'),nl,
        read(L),nl,
        write('Enter the index : '),
        read(ID),
        del(ID,L,NL),nl,
        write('New list is '),write(NL).


        del(X,[X|Tail],Tail).          %deletion of element

        del(X,[Y|Tail],[Y|Tail1]):-
                                del(X,Tail,Tail1).


/*

        del(ID,L,NL). is the predicate used.
        L is the list.
        ID is the index whose element is going to be deleted.
        NL is the new list.

*/
```

Output:-

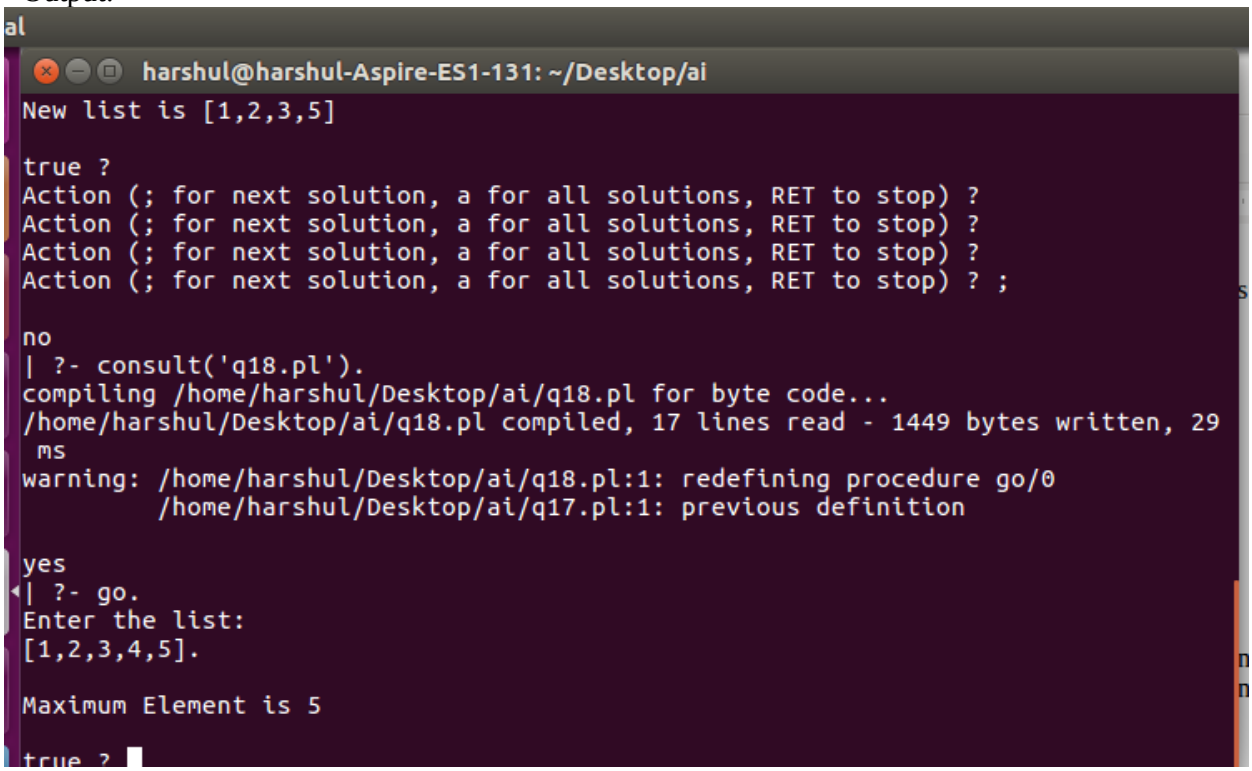


18. Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list L.

Code:-

```
go:-
        write('Enter the list:'),nl,
        read(L),nl,
        max(L,ME),
        write('Maximum Element is '),write(ME),nl.

max([H],H).
max([H|T],H):- max(T,K), H>=K.                %comparing condition
max([H|T],R):- max(T,R), R>H.                 %comparing condition

/*

        max(L,ME). is the predicate used.
        L is the list.
        ME is the maximum element of the list

*/
```

Output:-



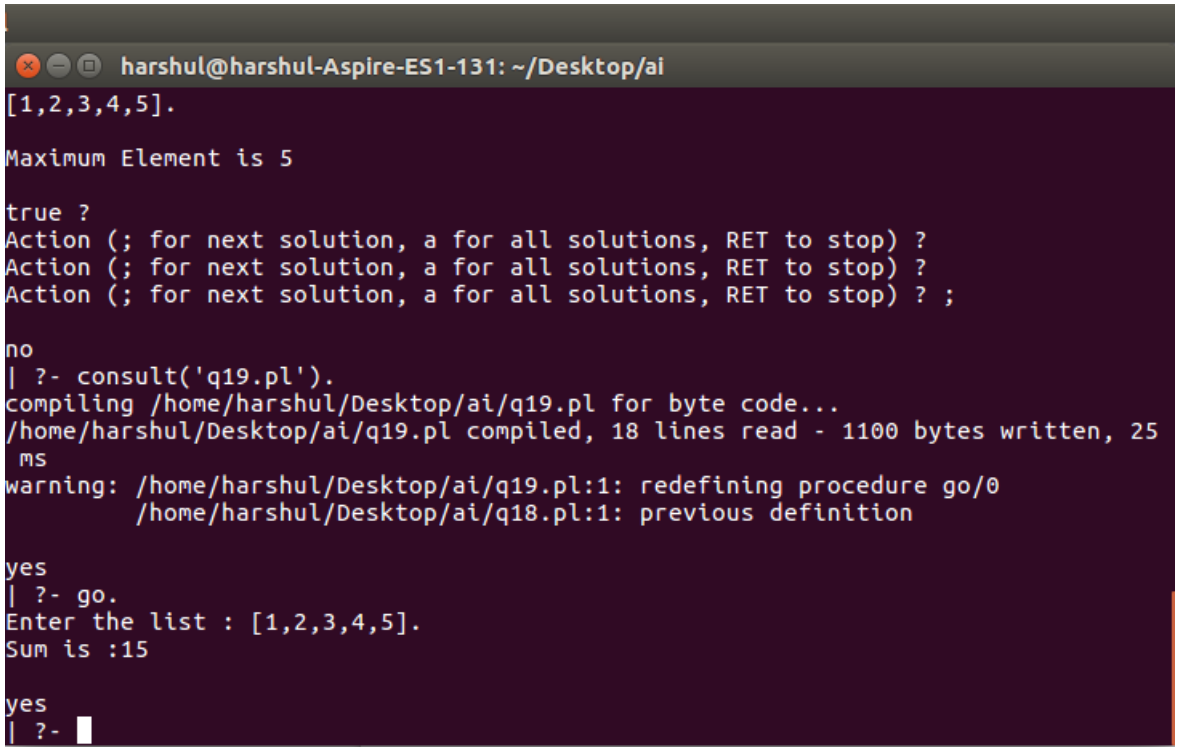
19. Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.20. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.

Code:-

```
go:-    write('Enter the list : '),
        read(N),
        write('Sum is :'),
        sum(N,K),
        write(K).

        sum([], 0).
        sum([H | T], TS) :-
                                sum(T, S1),
                                TS is H + S1.  %performing addition with every element of list
```

```
/*

        sum(N,K). is the predicate used.
        N is the list.
        K is the sum of the list.

*/
```

Output:-

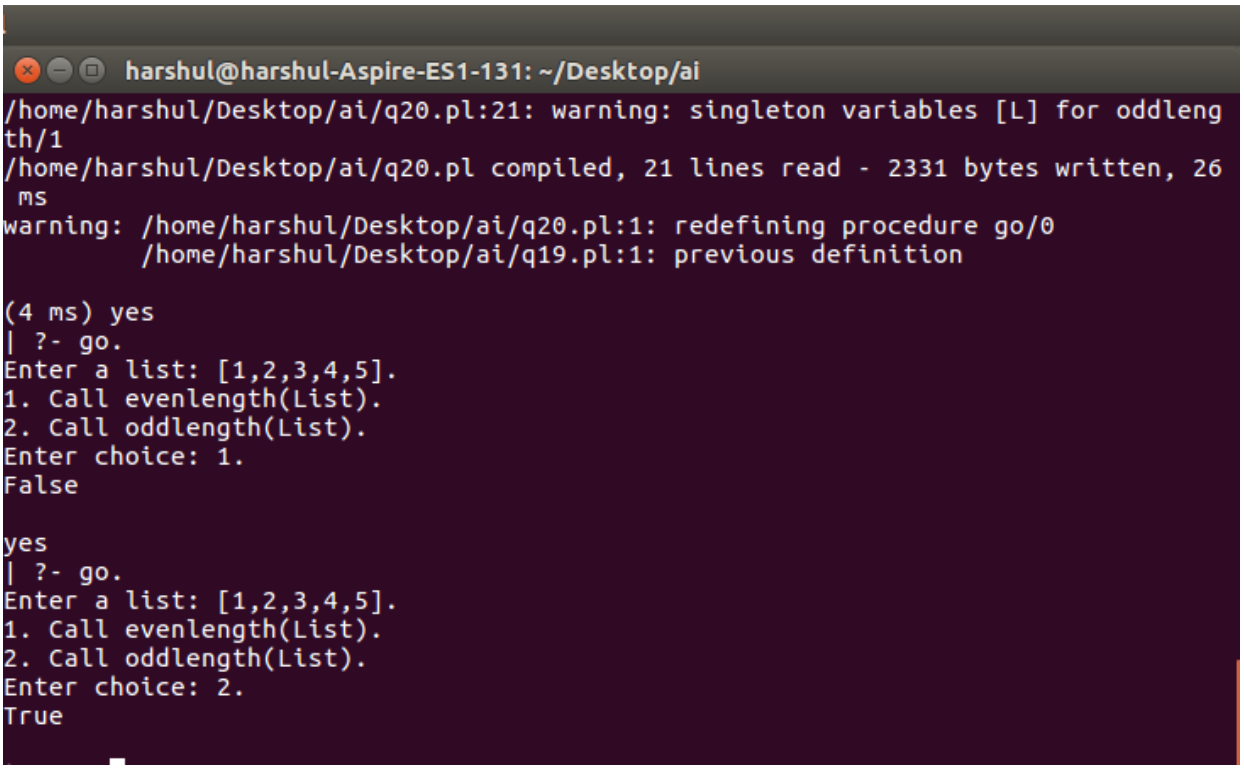


20. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.

Code:-

```
go:- write('Enter a list: '),
                    read(L),
                    write('1. Call evenlength(List).'),
                    nl,
                    write('2. Call oddlength(List).'),
                    nl,
                    write('Enter choice: '),
                    read(C),
                    fun(C,L),
                    nl.
fun(1,L) :- evenlength(L).
fun(2,L) :- oddlength(L).
evenlength(L) :- length(L,X),
                    R is X mod 2,
                     R==0, write('True'),!.
evenlength(L) :- write('False').
oddlength(L) :- length(L,X),
                     R is X mod 2,
                     R==1,
                    write('True').
oddlength(L) :- write('False').
```

Output:-

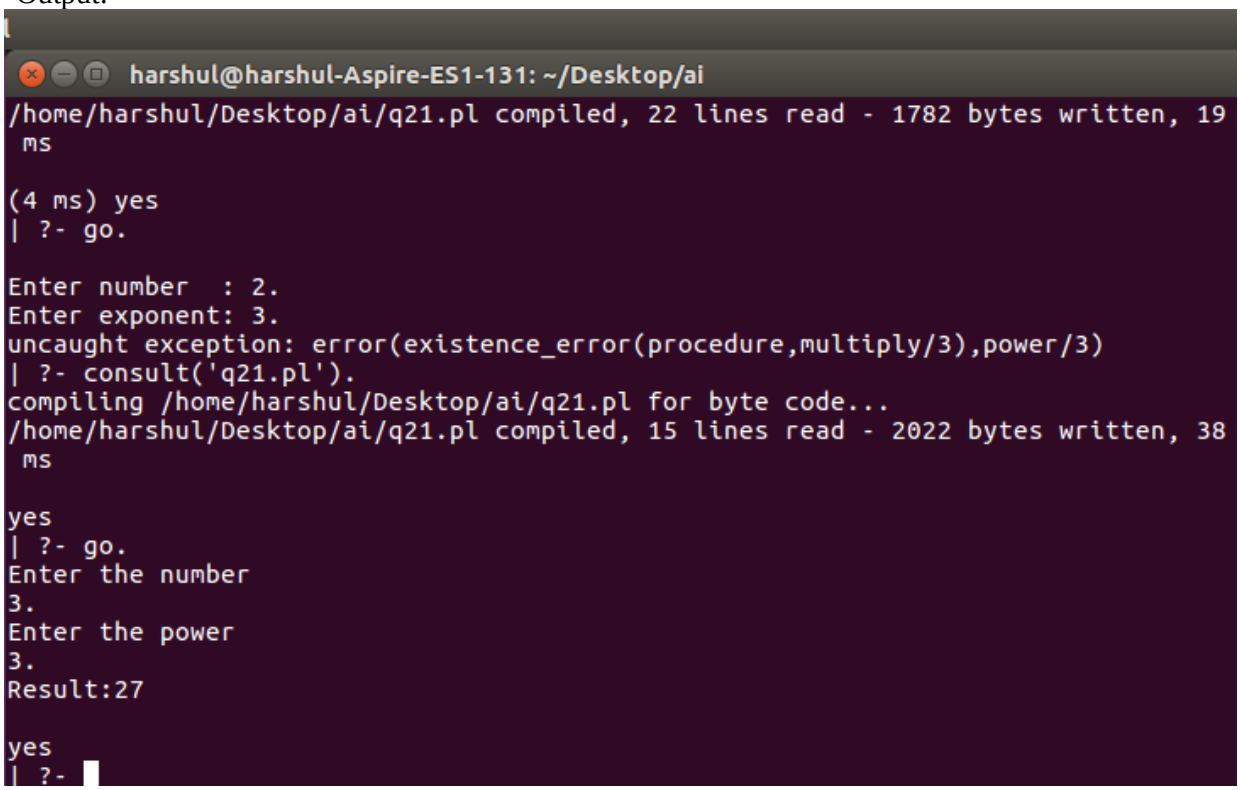


21. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.

Code:-

```
go:-
 nl,write('Enter number  : '),
 read(A),
 write('Enter exponent: '),
 read(B),
 power(A,B,X),
 write('Result       : '),
 write(X).

        power(X,0,1):-!.                %base condition forpower 0
        power(X,1,X):-!.                %base condition forpower 1
        power(X,Y,Res):- N is Y-1, multiply(X,N,R), Res is R*X.


/*

        power(A,B,X). is the predicate used.
        A is the Number.
        Bis the exponent power.
        X is the result.

*/
```

Output:-

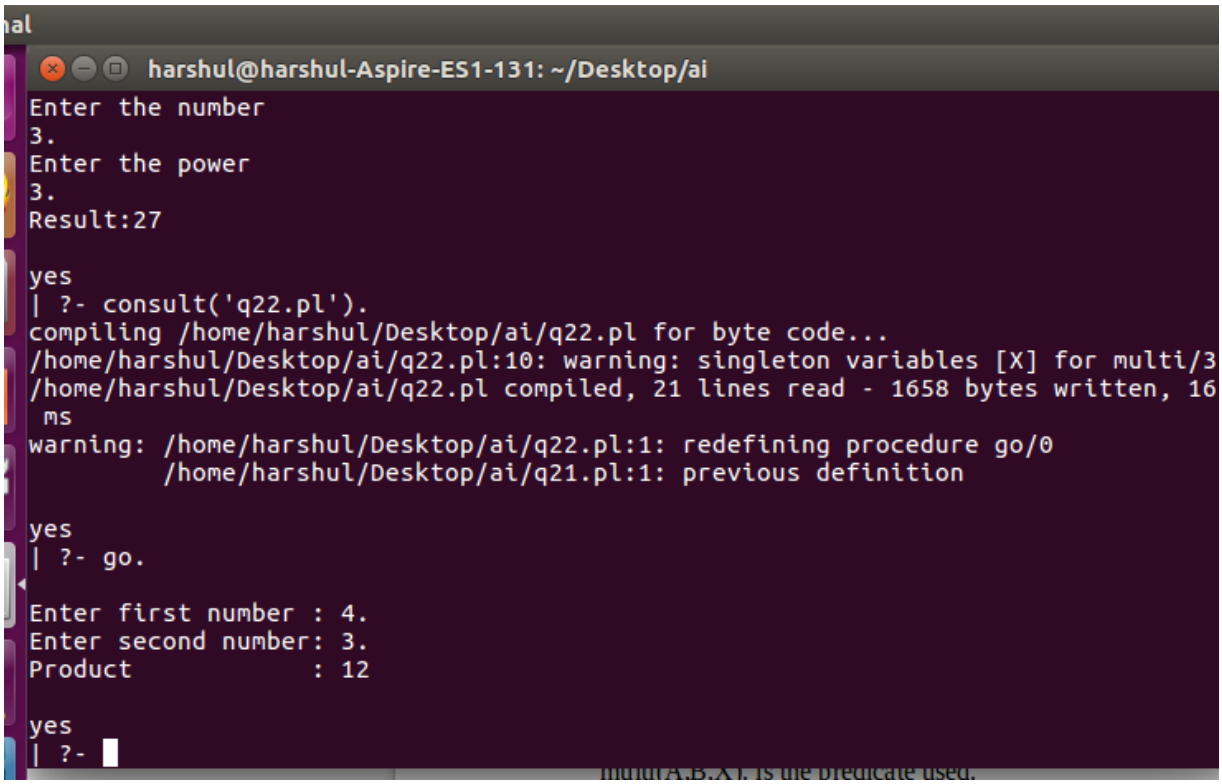


22. Write a Prolog program to implement multi (N1, N2, R) : where N1 and N2 denotes the numbers to be multiplied and R represents the result.

Code:-

```
go:-
 nl,write('Enter first number : '),
 read(A),
 write('Enter second number: '),
 read(B),
 multi(A,B,X),
 write('Product          : '),
 write(X).

multi(X,0,0):-!.                %base condtion ifsecond number is 0
multi(X,1,X):-!.                %base condtion ifsecond number is 1
multi(X,Y,Res):- N is Y-1, multi(X,N,R), Res is R+X.


/*
        multi(A,B,X). is the predicate used.
        A is the first number.
        B is the second number.
        X is the result of the multiplication.

*/
```

Output:-

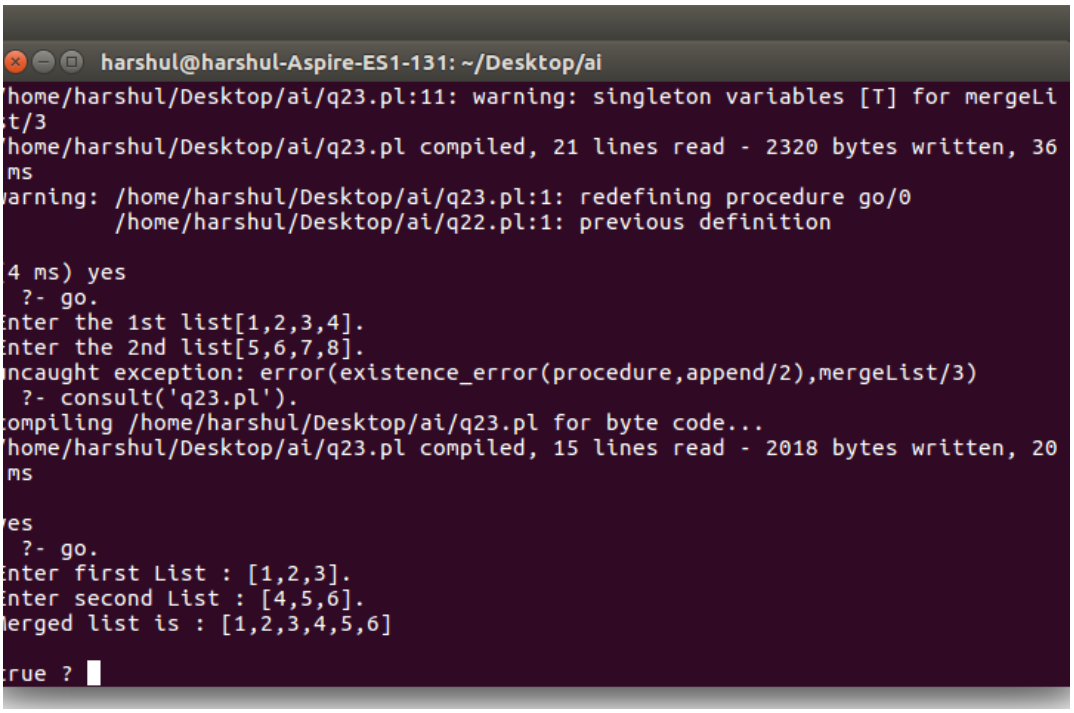


23. Write a program in PROLOG to implement merge (L1, L2, L3) where L1 is first ordered list and L2 is second ordered list and L3 represents the merged list.

Code:-

```
go:-    write('Enter the 1st list'),
        read(L1),
        write('Enter the 2nd list'),
        read(L2),
        mergeList(L1,L2,L),
        write(L).


        mergeList([X],[],[X]).
        mergeList([],[Y],[Y]).
        mergeList([X|L1],[Y|L2],[X|L]):- X=<Y,mergeList(L1,[Y|L2],L),append([X],T).
        mergeList([X|L1],[Y|L2],[Y|L]):- mergeList([X|L1],L2,L),append([Y]).

/*

        mergeList(L1,L2,L) is the predicate used.
        L1 is the first list.
        L2 is the second list.
        L is the merged list of L1 and L2.

*/
```

Output:-

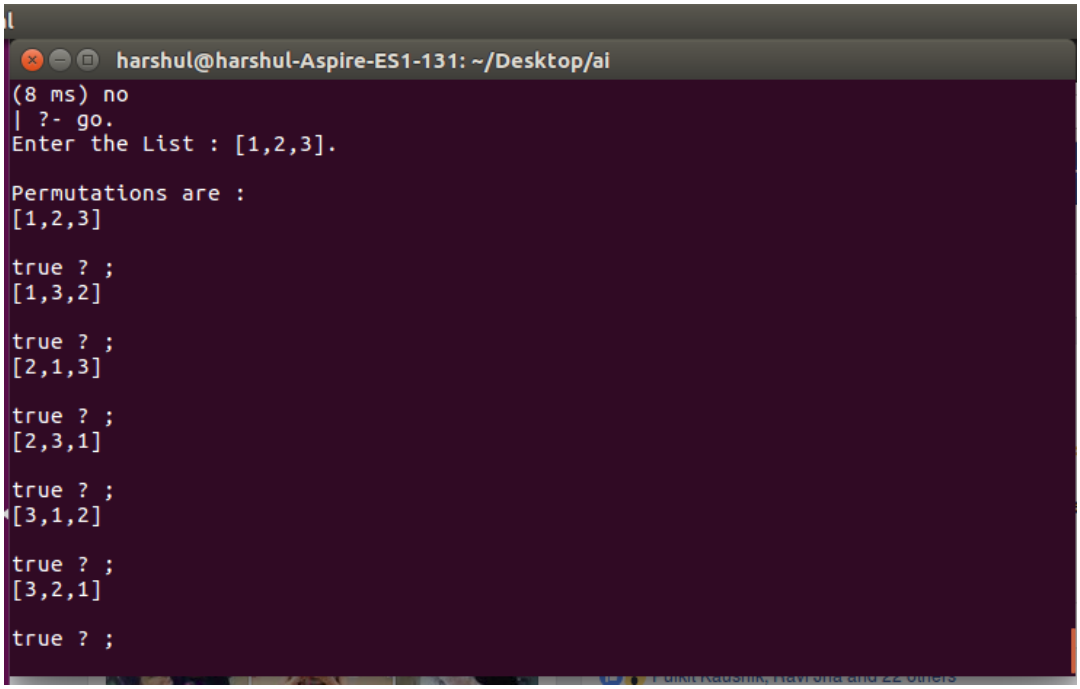


24. Write a program in PROLOG to implement permute (L, P) where P represents all possible permutations of the elements of List L.

Code:-

```
go:-
        write('Enter the List : '),
        read(L),nl,
        write('Permutations are : '),nl,
        permute(L,P),
        write(P).

        del(X,[X|L1],L1).                    %computing combinations
        del(X,[Y|L1],[Y|L2]):-
    del(X,L1,L2).

        permute([],[]).                      %base condition
        permute(L,[X|P]):-
        del(X,L,L1),
        permute(L1,P).


/*

        del(X,L,L1) and permute(L,P) are the predicates used.
        L is the list.
        P is the types of element arrangments that L can have.

*/
```

Output:-

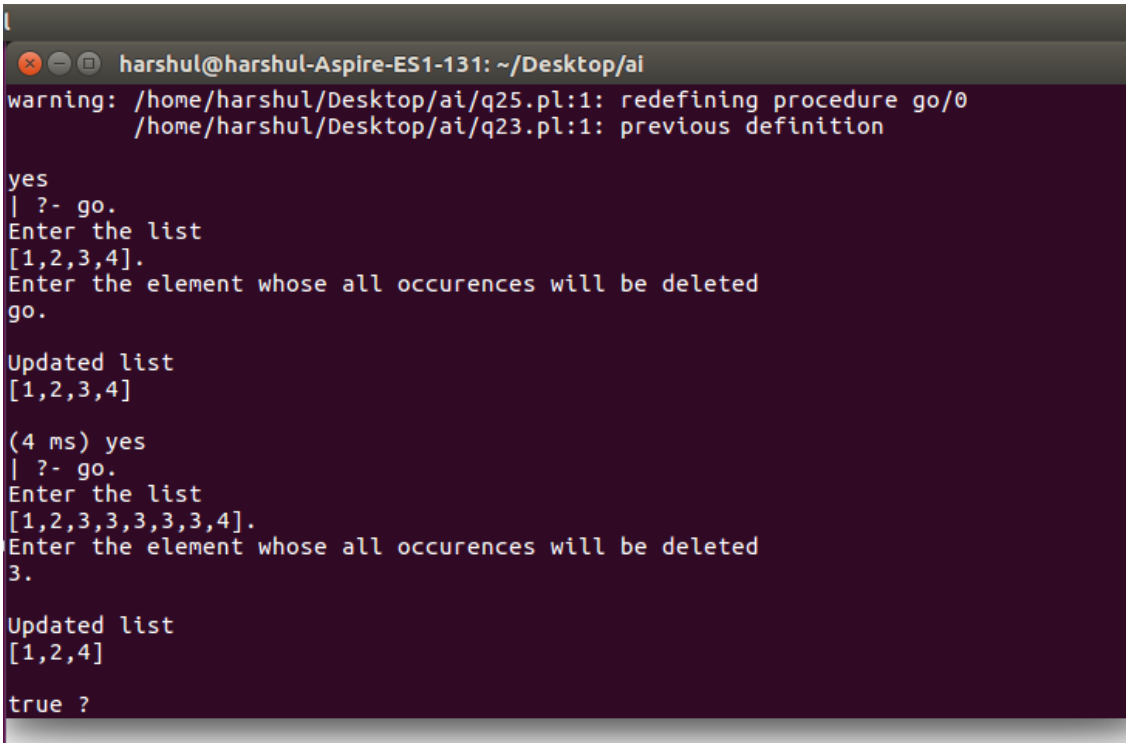


25. Write a program in PROLOG to implement delete_all (X, L, R) where X denotes the element whose all occurrences has to be deleted from list L to obtain list R.

Code:-

```
go:-    write('Enter the list'),
        nl,
        read(L),
        write('Enter the element whose all occurences will be deleted'),
        nl,
        read(X),nl,
        write('Updated list'),
        nl,
        delete_all(L,X,K),
        write(K).


        delete_all([],A,[]).
        delete_all([H|T],A,Result) :- H=A,
                                delete_all(T,A,Result).
        delete_all([H|T],A,[H|Result]) :- delete_all(T,A,Result).
```

Output:-

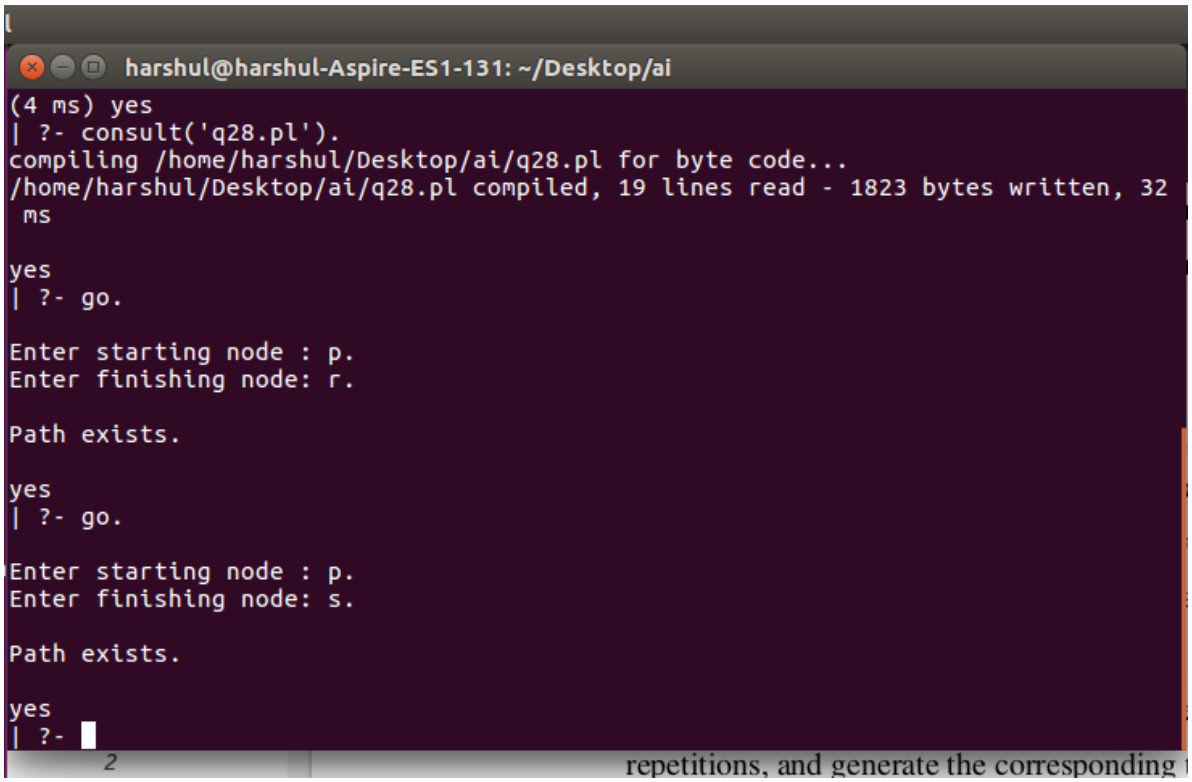


28. Consider a cyclic directed graph [edge (p, q), edge (q, r), edge (q, r), edge (q, s), edge (s,t)] where edge (A,B) is a predicate indicating directed edge in a graph from a node A to a node B. Write a program to check whether there is a route from one node to another node.

Code:-

```
go:-
   nl,write('Enter starting node : '),
   read(A),
   write('Enter finishing node: '),
   read(B),
   route(A,B),nl,
   write('Path exists.'),nl,!.

go:-
   nl,write('No path exists.'),nl.


route(A,B) :- edge(A,B),!.
route(A,B) :- edge(A,X), route(X,B).

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

Output:-



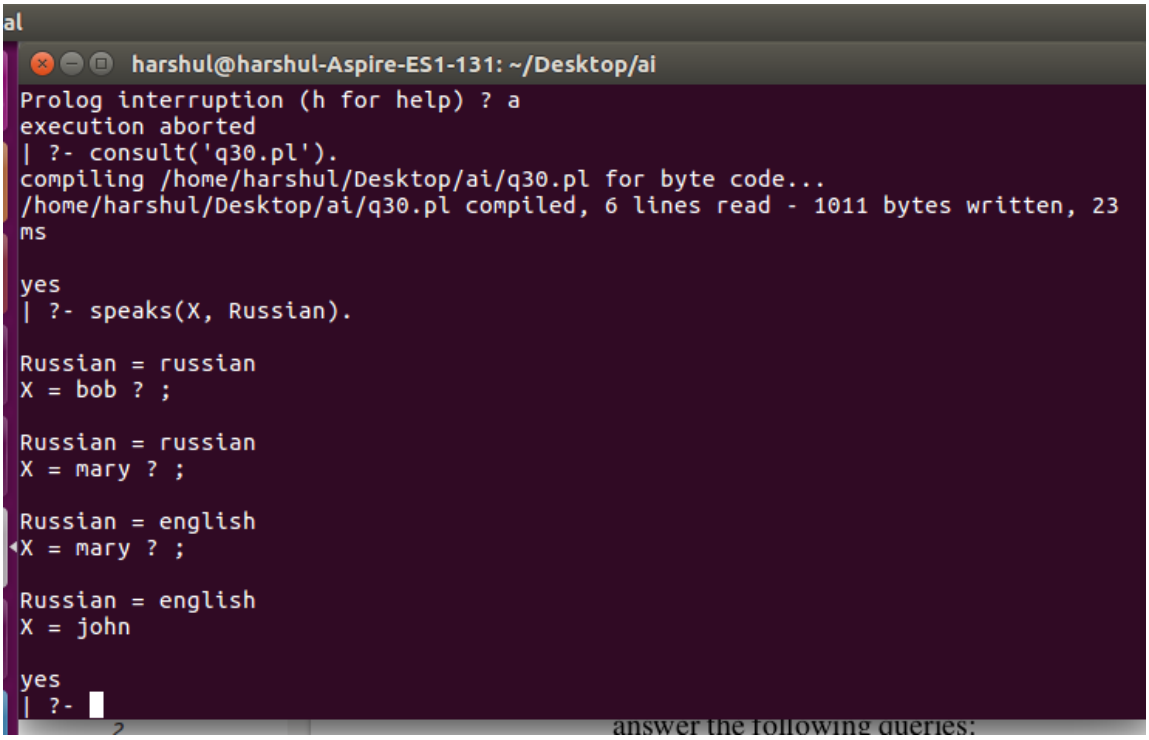
30. Using prolog, write a series of facts and rules that asserts the facts that Bob and Mary speak Russian and John and Mary speak English. It also defines the relation "understands" between two persons, which is true exactly when they both speak the same language. Your program should answer the following queries:
a) ?- speaks(X, Russian).
b) ?- understands(John, Bob).
c) ?- understands(X, Bob).
d) ?- understands(P1, P2).

Code:-

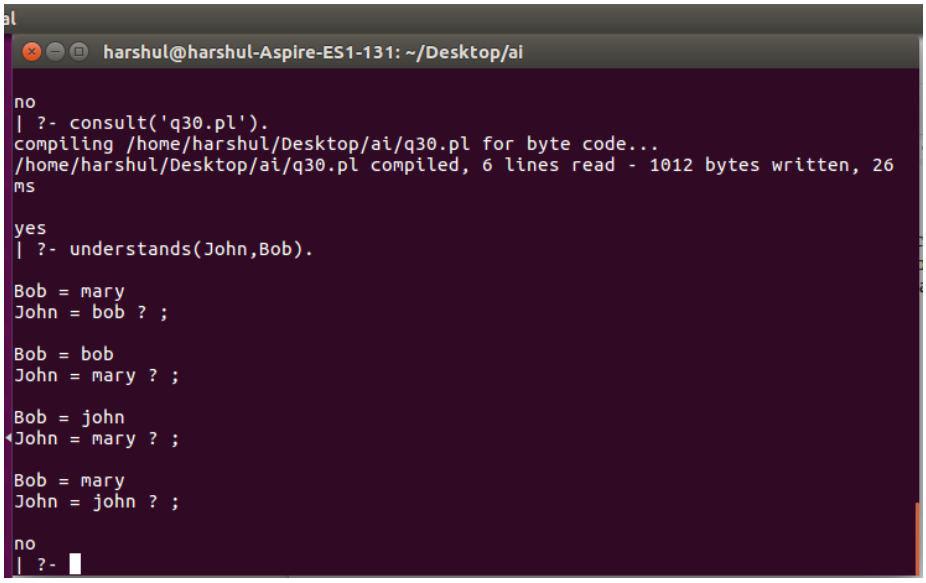
```
speaks(bob,russian).
speaks(mary,russian).
speaks(mary,english).
speaks(john,english).

understands(P1,P2):- speaks(P1,X),speaks(P2,X),P1\=P2.
```

Output:-

a)



b)


harshul@harshul-Aspire-ES1-131: ~/Desktop/ai
no
| ?- consult('q30.pl').
compiling /home/harshul/Desktop/ai/q30.pl for byte code...
/home/harshul/Desktop/ai/q30.pl compiled, 6 lines read - 1012 bytes written, 26 ms
yes
| ?- understands(John,Bob).
Bob = mary
John = bob ? ;
Bob = bob
John = mary ? ;
Bob = john
John = mary ? ;
Bob = mary
John = john ? ;
no
| ?-

c)


no
| ?- understands(X,Bob).
Bob = mary
X = bob ? ;
Bob = bob
X = mary ? ;
Bob = john
X = mary ? ;
Bob = mary
X = john ? ;
no
| ?-

d)


no
| ?- understands(P1,P2).
P1 = bob
P2 = mary ? ;
P1 = mary
P2 = bob ? ;
P1 = mary
P2 = john ? ;
P1 = john
P2 = mary ? ;
no
| ?-