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

go :-   write('First :'),

        read(A),

        write('Second :'),

        read(B),

        sum(A,B,R),

        write('Sum :'),

        write(R), nl.

sum(A,B,R) :- R is A+B.

:-initialization(go).

****

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

go :-   write('First :'),

        read(X),

        write('Second :'),

        read(Y),

        max(X,Y,M),

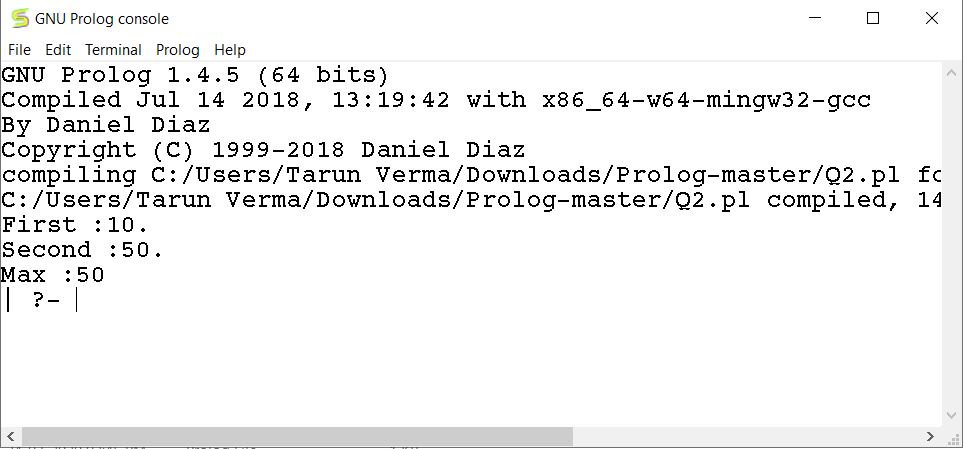
        write('Max :'),

        write(M), nl.

max(X,Y,Y) :- X<Y.

max(X,Y,X) :- X>Y.

:-initialization(go).

****

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

go :-   write('Enter Number :'),

        read(N),

        factorial(N,F),

        write('Factorial :'),

        write(F), nl.

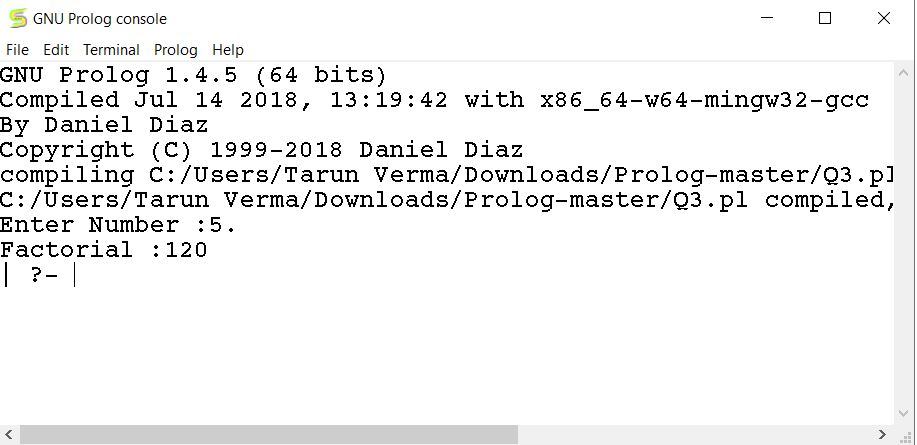
factorial(0,1).

factorial(N,X):-N1 is N-1,

                factorial(N1,X1),

                X is X1\*N.

:-initialization(go).

****

**4. Write a program in PROLOG to implement generate\_fib(N,T) where T represents the Nth term of the Fibonacci series.**

go :-   write('Enter Number :'),

        read(N),

        generate\_fib(N-1,T),

        write('Result :'),

        write(T), nl.

generate\_fib(0,1).

generate\_fib(1,1).

generate\_fib(N,T):- N1 is N-1,

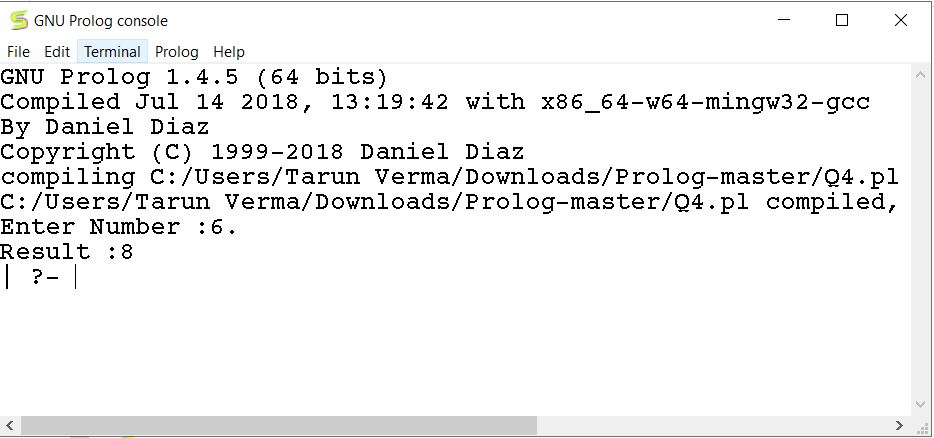
                    generate\_fib(N1,T1),

                    N2 is N-2,

                    generate\_fib(N2,T2),

                    T is T1+T2.

:-initialization(go).

****

**5. Write a Prolog program to implement GCD of two numbers.**

go :-   write('First :'),

        read(A),

        write('Second :'),

        read(B),

        gcd(A,B,R),

        write('Result :'),

        write(R), nl.

gcd(X,X,X).

gcd(X,Y,D):- X<Y,

             gcd(Y,X,D).

gcd(X,Y,D):- X>Y,

             Y1 is X-Y,

             gcd(Y,Y1,D).

:-initialization(go).

****

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

go :-   write('Number :'),

        read(Num),

        write('Power :'),

        read(Pow),

        power(Num,Pow, Ans),

        write('Result :'),

        write(Ans), nl.

power(Num,1,Num).

power(Num,Pow,Ans):- Pow1 is Pow-1,

                    power(Num,Pow1,Ans1),

                    Ans is Ans1\*Num.

:-initialization(go).

****

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

go :-   write('First :'),

        read(N1),

        write('Second :'),

        read(N2),

        multi(N1,N2,R),

        write('Sum :'),

        write(R), nl.

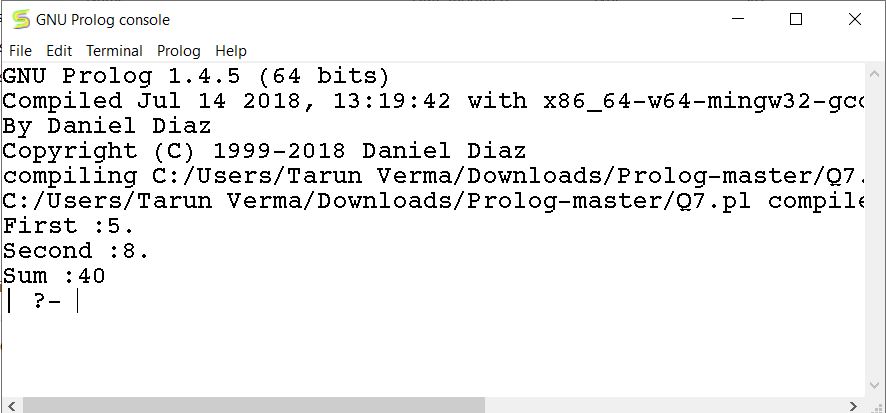
multi(N1,1,N1).

multi(N1,N2,R):-  Temp is N2-1,

                  multi(N1,Temp,R1),

                  R is R1+N1.

:-initialization(go).

****

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

go :-   write('Enter Number :'),

        read(N),

        towerofhanoi(N,T),

        write('Result :'),

        write(T), nl.

power(Num,1,Num).

power(Num,Pow,Ans):- Pow1 is Pow-1,

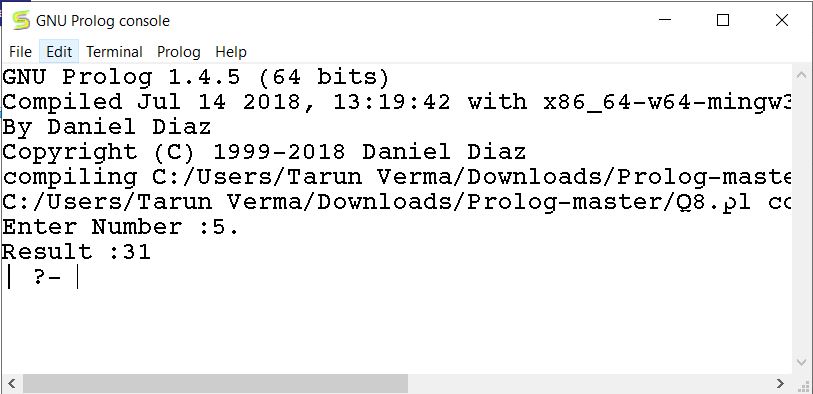
                     power(Num,Pow1,Ans1),

                     Ans is Ans1\*Num.

towerofhanoi(N,Ans):- power(2,N,Ans1),

             Ans is Ans1-1.

:-initialization(go).

****

**9. 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.**

edge(p,q).

edge(q,r).

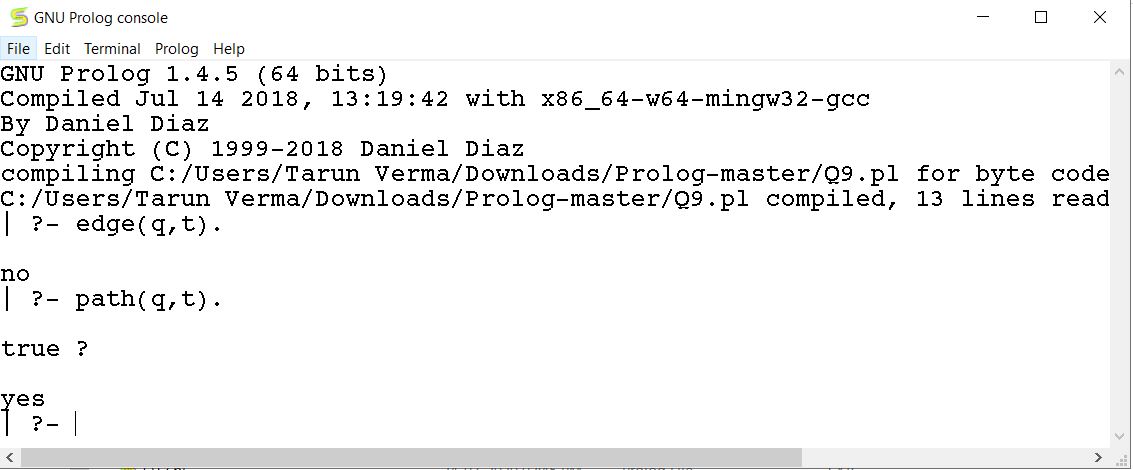
edge(q,s).

edge(s,t).

path(X,Y) :- edge(X,Y).

path(X,Y) :- edge(X,Z),

            path(Z,Y).

****

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

go :- write('Enter a list'), nl,

      createList(L1),

      write('Enter element to find :'),

      read(X),

      memb(X,L1), nl.

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

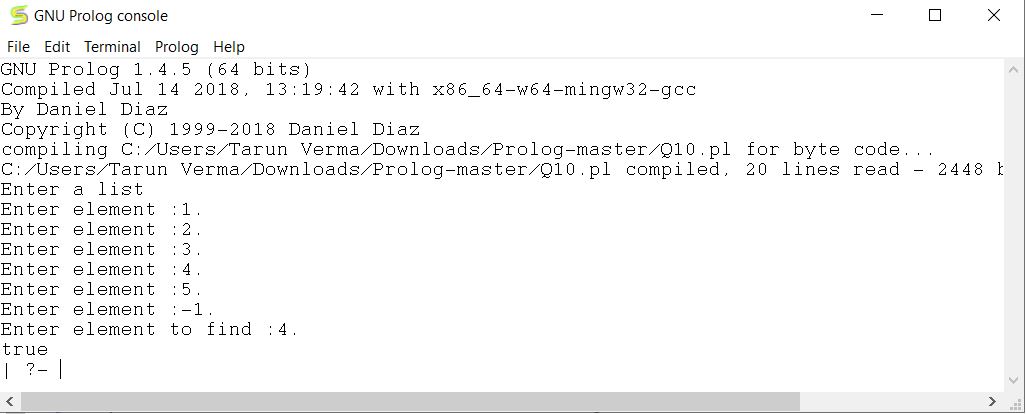
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

memb(\_,[]) :- write('false'),!.

memb(X,[X|\_]) :- write('true'),!.

memb(X,[\_|T]) :- memb(X,T).

:-initialization(go).

****

**11. 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.**

go :- write('Enter a list 1'), nl,

      createList(L1),

      write('Enter a list 2'), nl,

      createList(L2),

      write("Appending List ..."),

      conc(L1,L2,L3),

      printList(L3), nl.

conc([],L2,L2).

conc([H|T1],L2,[H|T2]):- conc(T1,L2,T2).

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

enterEle(X) :- write('Enter element :'),

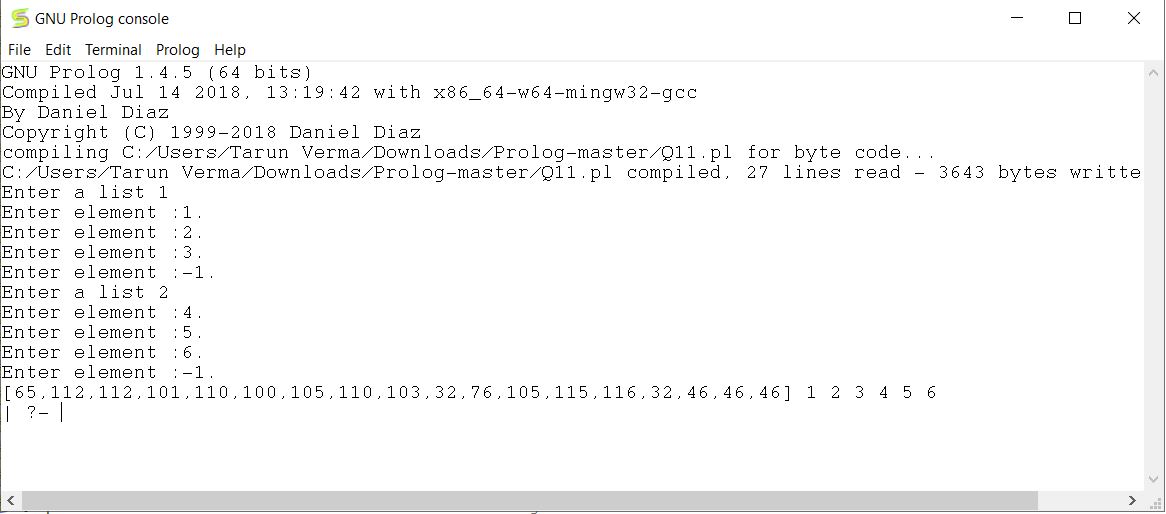
               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

:-initialization(go).

****

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

go :- write('Enter a list'), nl,

      createList(L),

      write(" "),

      myReverse(L,R),nl,nl,

      write('Reversed List : '),

      printList(R), nl,nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

enterEle(X) :- write('Enter element :'),

               read(X).

conc([],L2,L2).

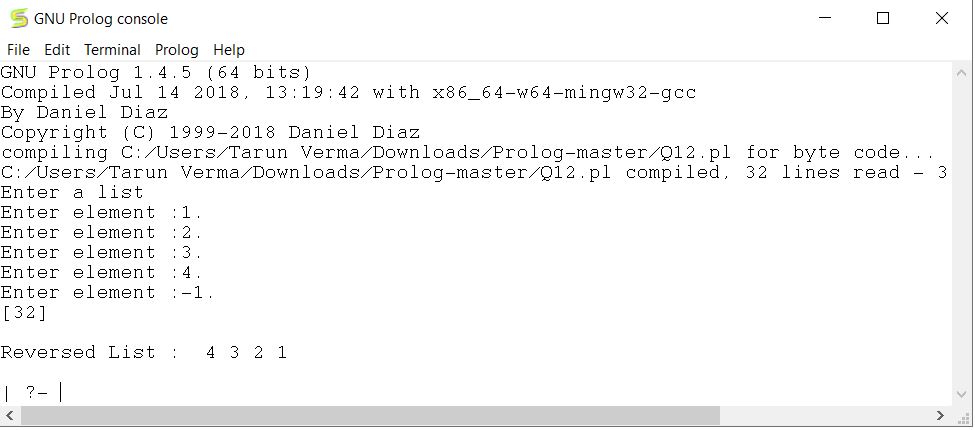
conc([H|T1],L2,[H|T2]):- conc(T1,L2,T2).

myReverse([],[]).

myReverse([H|T],R):- myReverse(T,R1),

                conc(R1,[H],R).

:-initialization(go).

****

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

go :- write('Enter a list'), nl,

      createList(L),

      palindrome(L), nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

enterEle(X) :- write('Enter element :'),

               read(X).

conc([],L2,L2).

conc([H|T1],L2,[H|T2]):- conc(T1,L2,T2).

myReverse([],[]).

myReverse([H|T],R):- myReverse(T,R1),

                conc(R1,[H],R).

isEqual([],[]):- write('true').

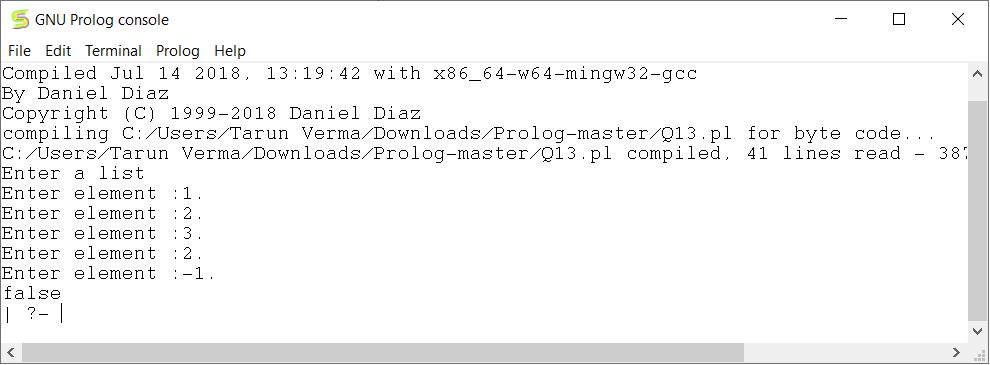
isEqual([\_|\_],[\_|\_]):- write('false').

isEqual([H1|T1],[H1|T2]):- isEqual(T1,T2).

palindrome(L):- myReverse(L,R),

                isEqual(L,R).

:-initialization(go).

****

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

go :- write('Enter a list'), nl,

      createList(L1),

      write('SumList :'),

      sumlist(L1,S),

      write(S), nl.

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

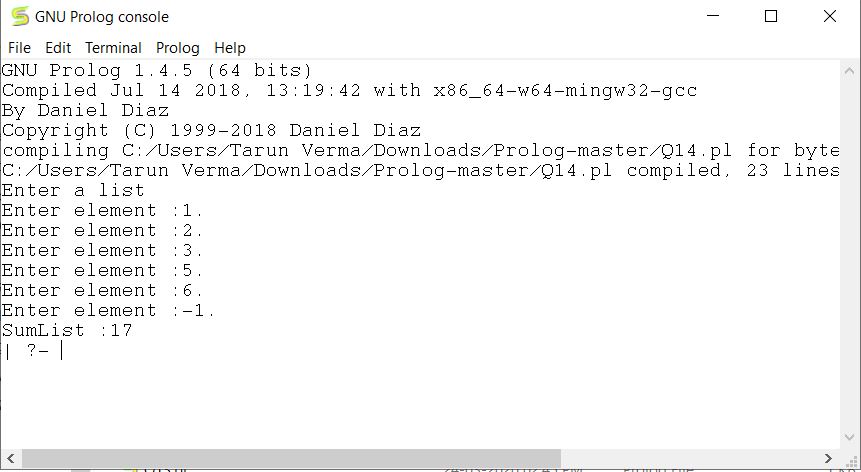
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

sumlist([],0).

sumlist([H|T],S):- sumlist(T,S1),

                   S is H+S1.

:-initialization(go).

****

**15. 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**

go :-

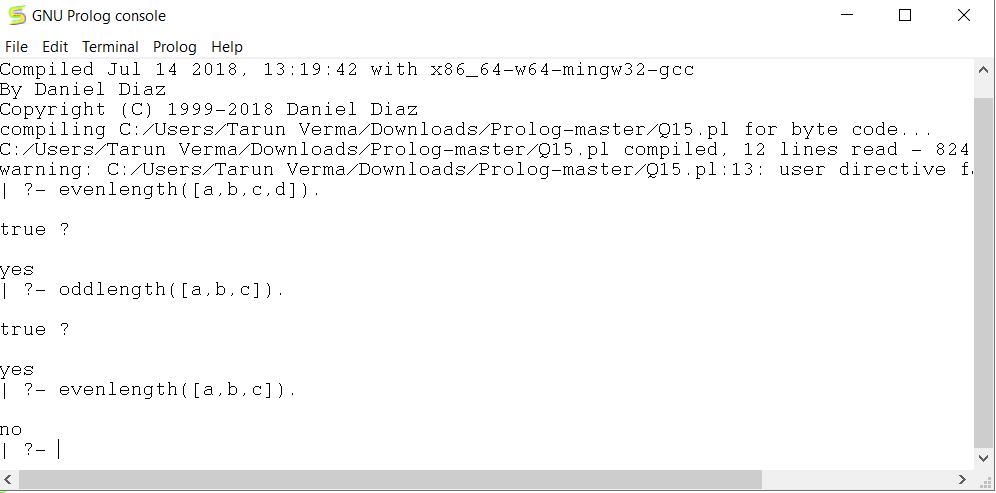
evenlength([]).

evenlength([\_|T]):- oddlength(T).

oddlength([\_]).

oddlength([\_|T]) :- evenlength(T).

:-initialization(go).

****

**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.**

go :- write('Enter a list'), nl,

      createList(L),

      write('Enter the index :'),

      read(N),

      nth\_element(N,L,X),

      write('Element :'),

      write(X), nl.

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

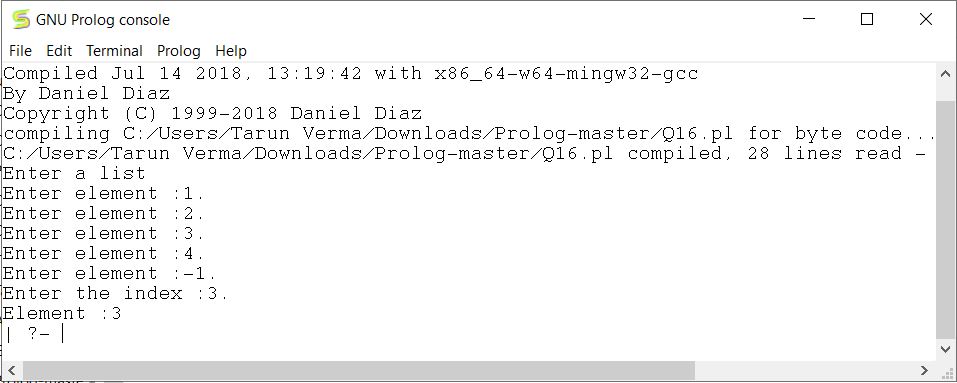
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

nth\_element(1,[H|\_],H).

nth\_element(N,[\_|T],X):- N1 is N-1,

                        nth\_element(N1,T,X).

:-initialization(go).

****

**17. 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.**

go:- write('Enter a list'),nl,

     createList(L),nl,

     remove\_dup(L,R),

     write('Modified List : '),

     printList(R), nl.

memb([],\_):-    !,fail.

memb([X|\_],X):- !.

memb([\_|T],Y):- memb(T,Y).

remove\_dup([],[]).

remove\_dup([X|T],T1):-  memb(T,X),!,

                        remove\_dup(T,T1).

remove\_dup([X|T],[X|T1]):- remove\_dup(T,T1).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

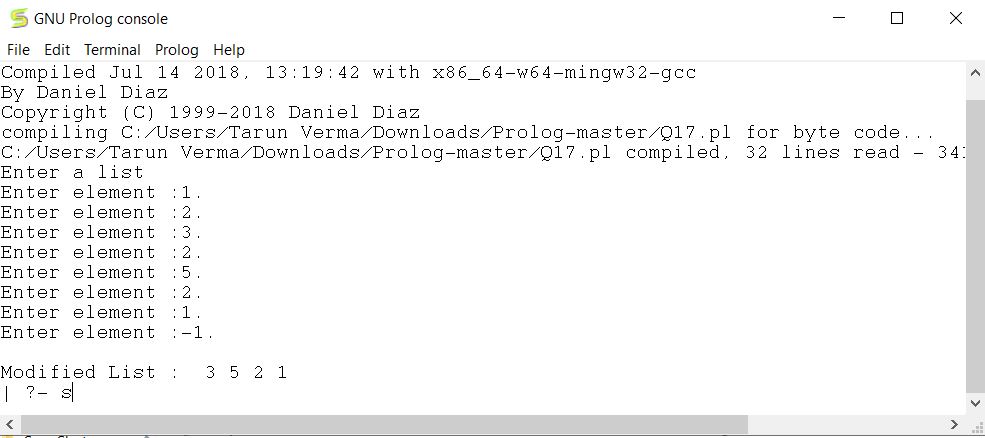
createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

:-initialization(go).

****

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

go :- write('Enter a list'), nl,

      createList(L),

      maxlist(L,M),

      write('Maximum Number :'),

      write(M), nl.

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

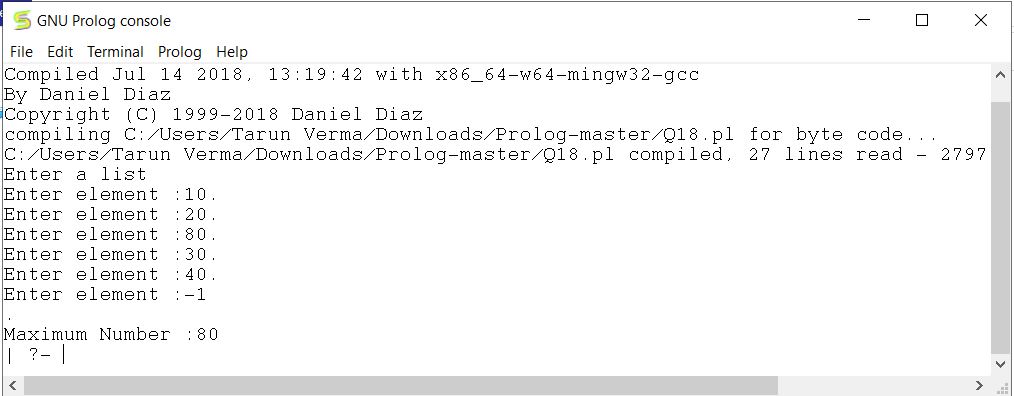
maxlist([H],H).

maxlist([H|T],M):- maxlist(T,M1),

                H<M1 -> M is M1;

                M is H.

:-initialization(go).

****

**19. 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.**

go :- write('Enter a list'), nl,

      createList(L),

      write('Enter the index :'),

      read(N),

      write('Enter Element :'),

      read(I),

      insert\_nth(I,N,L,R),

      write('Modified List :'),

      printList(R), nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

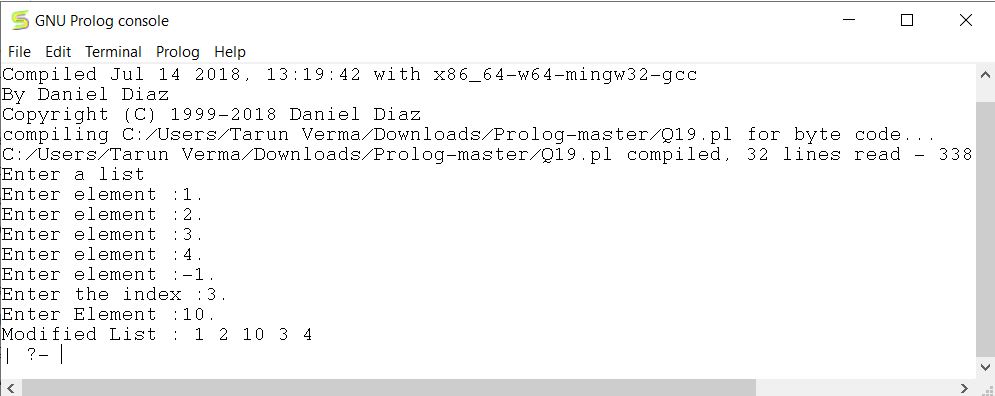
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

insert\_nth(I, 1, L, [I|L]).

insert\_nth(I, N, [H|T], [H|R]):- N1 is N-1,

                                insert\_nth(I, N1, T, R).

:-initialization(go).

****

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

go :- write('Enter a list'), nl,

      createList(L),

      write('Enter the sublist :'), nl,

      createList(S),

      mysublist(S,L).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

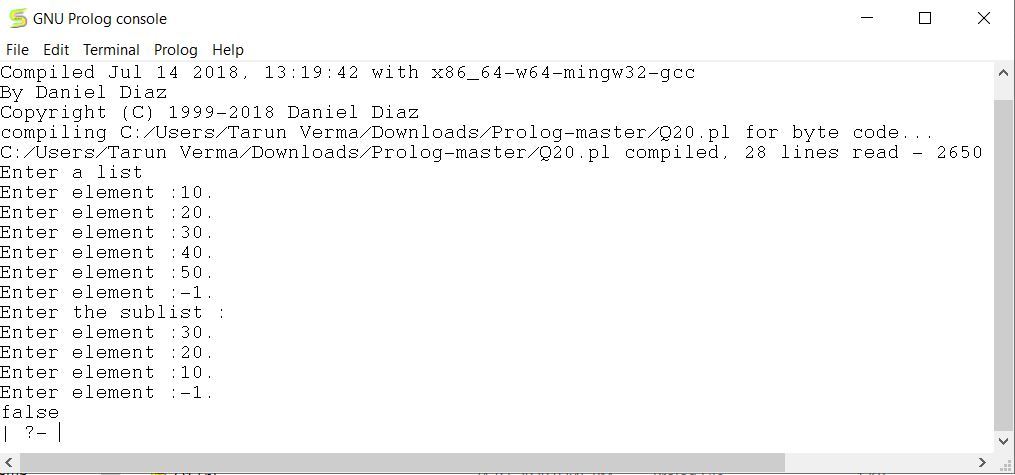
mysublist([],\_).

mysublist(\_,[]):- write('false'), nl.

mysublist([H1|T1],[H1|T2]):- mysublist(T1,T2).

mysublist([H1|T1],[\_|T2]):- mysublist([H1|T1],T2).

:-initialization(go).

****

**21. 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.**

go :- write('Enter a list'), nl,

      createList(L),

      write('Enter the index :'),

      read(N),

      delete\_nth(N,L,R),

      write('Modified List :'),

      printList(R), nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

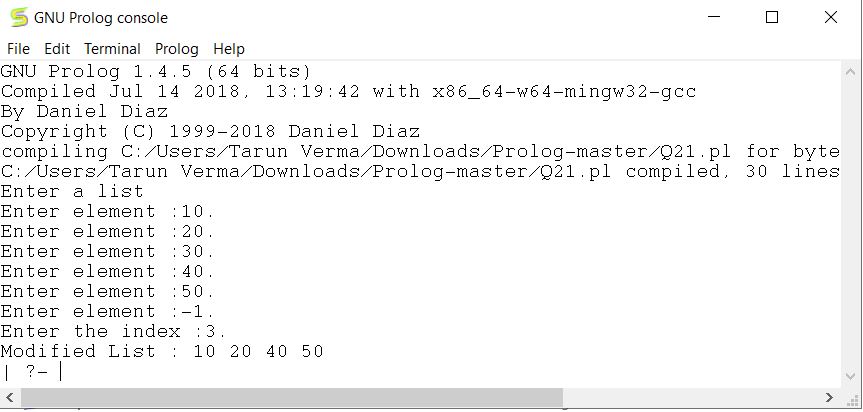
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

delete\_nth(1, [\_|T], T).

delete\_nth(N, [H|T], [H|R]):- N1 is N-1,

                            delete\_nth(N1, T, R).

:-initialization(go).

****

**22. 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.**

go :- write('Enter a list'), nl,

      createList(L),

      write('Enter the element :'),

      read(X),

      delete\_all(X,L,R),

      write('Modified List :'),

      printList(R), nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

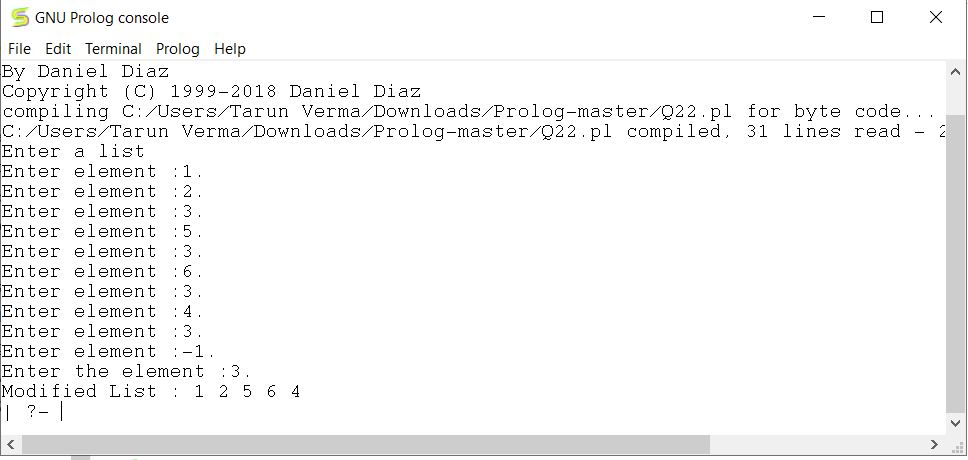
createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

delete\_all(\_, [], []).

delete\_all(X, [X|T], R):- delete\_all(X, T, R).

delete\_all(X, [H|T], [H|R]):- delete\_all(X, T, R).

:-initialization(go).

****

**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.**

go :- write('Enter list 1'), nl,

      createList(L1),

      write('Enter list 2 :'),

      createList(L2), nl,

      myMerge(L1,L2,L3),

      write('Merged List :'),

      printList(L3), nl.

printList([]).

printList([H|T]) :- write(' '), write(H), printList(T).

enterEle(X) :- write('Enter element :'),

               read(X).

createList(L1) :- enterEle(X), createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1), createList(X1,T).

myMerge([],L2,L2).

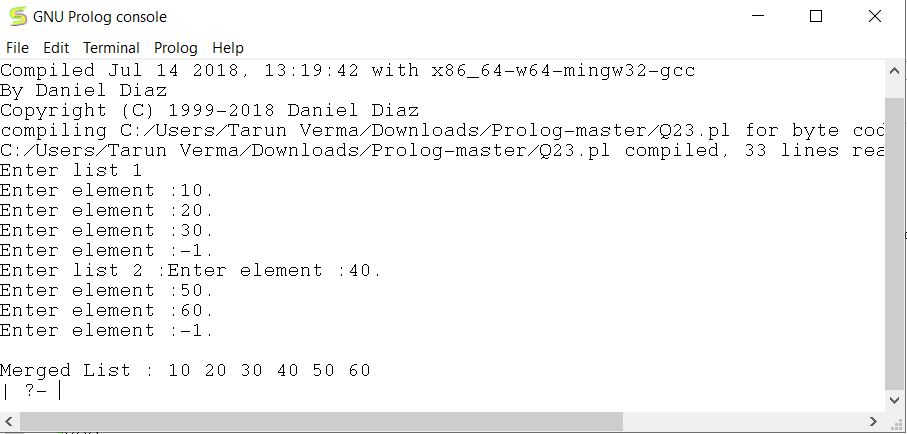
myMerge(L1,[],L1).

myMerge([H1|T1],[H2|T2],[H1|T3]):- H1=<H2,

                                myMerge(T1, [H2|T2], T3).

myMerge([H1|T1],[H2|T2],[H2|T3]):- myMerge([H1|T1], T2, T3).

:-initialization(go).

****

**24. Write a program in prolog for calculating the length of list.**

go :- write('Enter a list: '),nl,

      createList(L),

      length\_of(L,N),

      write('Length : '),

      R is N,

      write(R), nl.

enterEle(X) :- write('Enter element:- '),

               read(X).

createList(L1) :- enterEle(X),createList(X,L1).

createList(-1,[]) :- !.

createList(X,[X|T]) :- enterEle(X1) , createList(X1,T).

length\_of([],0).

length\_of([\_|T],N) :- N = N1+1 , length\_of(T , N1).

:-initialization(go).

