Name of the program: B.Sc. (H) Computer Science

Semester: VI

Paper Name: Artificial Intelligence

Paper Code: 32341601

Name: KANIKA GUPTA

College Roll Number: 2019321

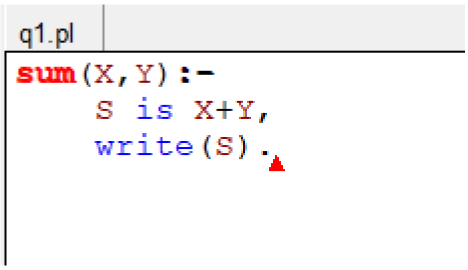
Examination Roll Number: 19066570019

Date: 24.04.2022

Q1. Write a prolog program to calculate the sum of two numbers.

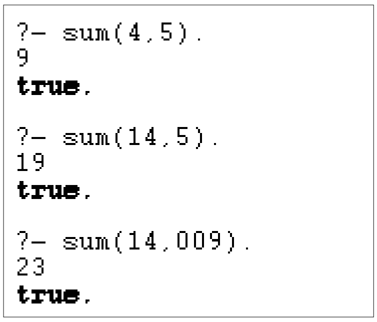
CODE

```
sum(X,Y):-
    S is X+Y,
    write(S).
```



```
sum(X,Y):-
      S is X+Y,
      write(S).
```

OUTPUT

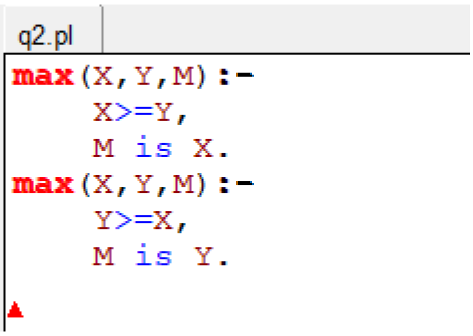


```
?- sum(4,5).
9
true.

?- sum(14,5).
19
true.

?- sum(14,009).
23
true.
```

Q2. Write a Prolog program to implement max(X, Y, M) so that M is the maximum of two numbers X and Y.

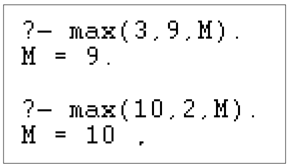
CODE

```
max(X,Y,M):-
   X>=Y,
   M is X.
max(X,Y,M):-
   Y>=X,
   M is Y.
```



```
max(X,Y,M):-
      X>=Y,
      M is X.
max(X,Y,M):-
      Y>=X,
      M is Y.
```

OUTPUT

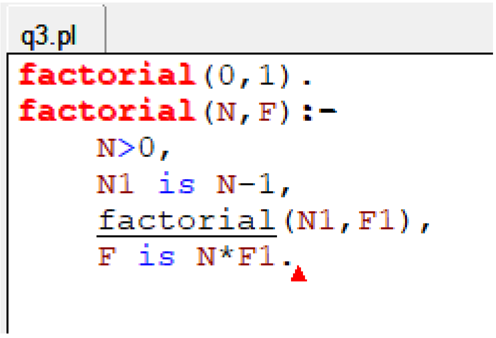


```
?- max(3,9,M).
M = 9.

?- max(10,2,M).
M = 10 .
```

Q3. Write a program in PROLOG to implement factorial (N, F) where F represents the factorial of a number N.

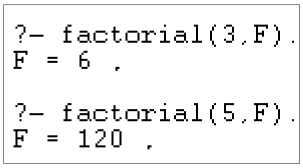
CODE

```
factorial(0,1).

factorial(N,F):-

   N>0,

   N1 is N-1,

   factorial(N1,F1),

   F is N*F1.
```



```
factorial(0,1).
factorial(N,F):-
      N>0,
      N1 is N-1,
      factorial(N1,F1),
      F is N*F1.
```

OUTPUT

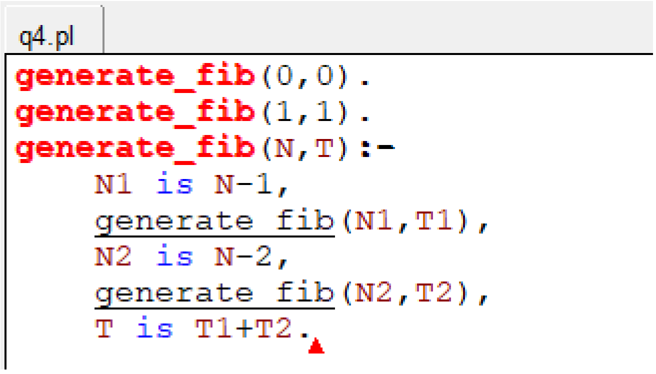


```
?- factorial(3,F).
F = 6 .

?- factorial(5,F).
F = 120 .
```

Q4. Write a program in PROLOG to implement generate_fib(N,T) where T represents the Nth term of the fibonacci series.

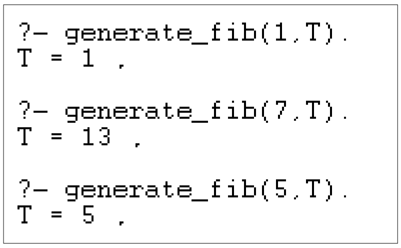
CODE

```
generate_fib(0,0).

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.
```



```
generate_fib(0,0).
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.
```

OUTPUT

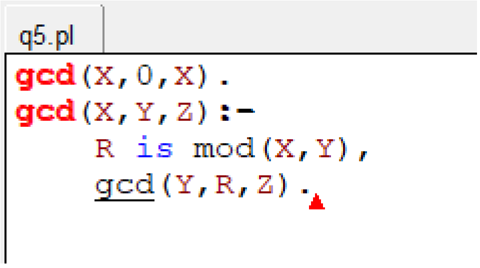


```
?- generate_fib(1,T).
T = 1 .

?- generate_fib(7,T).
T = 13 .

?- generate_fib(5,T).
T = 5 .
```

Q5. Write a Prolog program to implement GCD of two numbers.

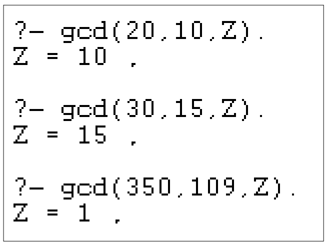
CODE

```
gcd(X,0,X).
gcd(X,Y,Z):-
   R is mod(X,Y),
   gcd(Y,R,Z).
```



```
gcd(X,0,X).
gcd(X,Y,Z):-
     R is mod(X,Y),
     gcd(Y,R,Z).
```

OUTPUT

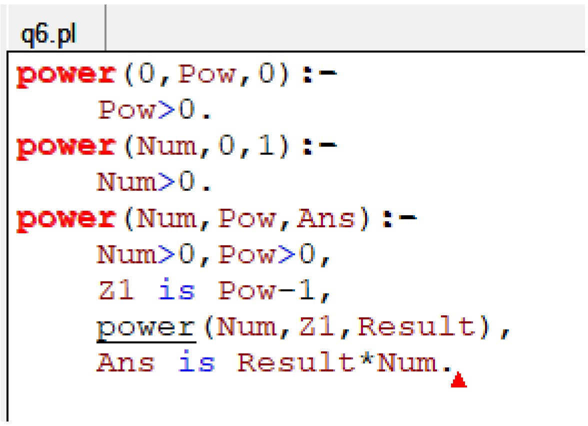


```
?- gcd(20,10,Z).
Z = 10 .

?- gcd(30,15,Z).
Z = 15 .

?- gcd(350,109,Z).
Z = 1 .
```

Q6. Write a Prolog program to implement power (Num,Pow, Ans) : where Num is raised to the power Pow to get Ans.

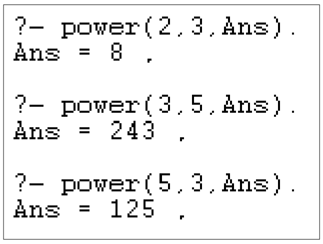
CODE

```
power(0,Pow,0):-
   Pow>0.
power(Num,0,1):-
   Num>0.
power(Num,Pow,Ans):-
   Num>0,Pow>0,
   Z1 is Pow-1,
   power(Num,Z1,Result),
   Ans is Result*Num.
```



```
power(0,Pow,0):-
      Pow>0.
power(Num,0,1):-
      Num>0.
power(Num,Pow,Ans):-
      Num>0,Pow>0,
      Z1 is Pow-1,
      power(Num,Z1,Result),
      Ans is Result*Num.
```

OUTPUT

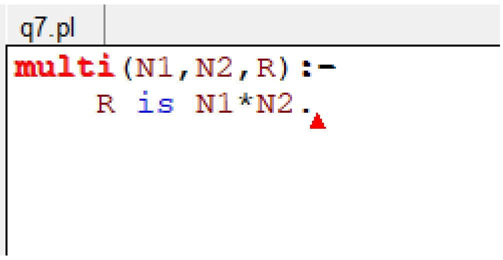


```
?- power(2,3,Ans).
Ans = 8 .

?- power(3,5,Ans).
Ans = 243 .

?- power(5,3,Ans).
Ans = 125 .
```

Q7. Prolog program to implement multi (N1, N2, R): where N1 and N2 denotes the numbers to be multiplied and R represents the result.

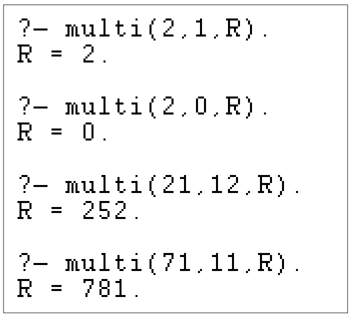
CODE

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



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

OUTPUT

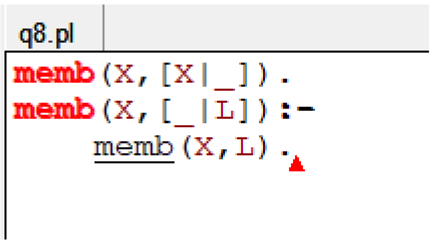


```
?- multi(2,1,R).
R = 2.

?- multi(2,0,R).
R = 0.

?- multi(21,12,R).
R = 252.

?- multi(71,11,R).
R = 781.
```

Q8. Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not

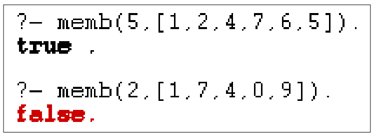
CODE

```
memb(X,[X|_]).

memb(X,[_|L]):-

   memb(X,L).
```



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

OUTPUT

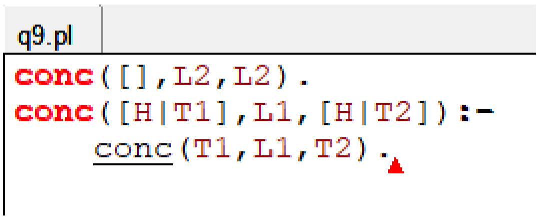


```
?- memb(5,[1,2,4,7,6,5]).
true .

?- memb(2,[1,7,4,0,9]).
false.
```

Q9. 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.

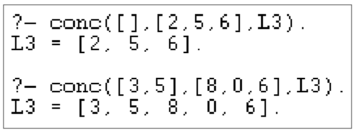
CODE

```
conc([],L2,L2).

conc([H|T1],L1,[H|T2]):-

   conc(T1,L1,T2).
```



```
conc([],L2,L2).
conc([H|T1],L1,[H|T2]):-
    conc(T1,L1,T2).
```

OUTPUT

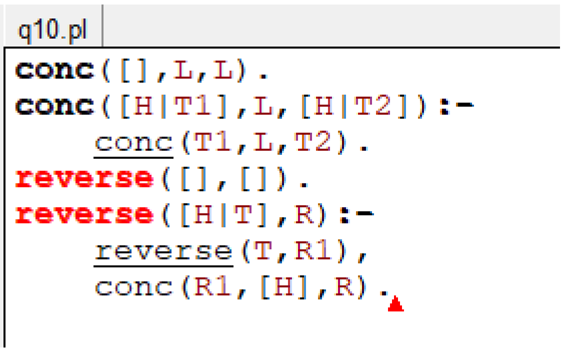


```
?- conc([],[2,5,6],L3).
L3 = [2, 5, 6].

?- conc([3,5],[8,0,6],L3).
L3 = [3, 5, 8, 0, 6].
```

Q10. Write a Prolog program to implement reverse (L, R) where List L is original and List R is reversed list.

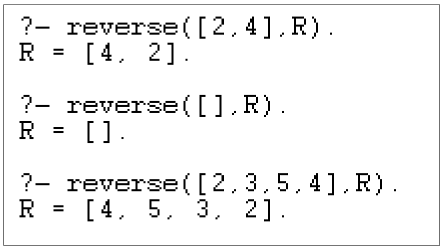
CODE

```
conc([],L,L).

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

   conc(T1,L,T2).

reverse([],[]).

reverse([H|T],R):-

   reverse(T,R1),

   conc(R1,[H],R).
```



```
conc([],L,L).
conc([H|T1],L,[H|T2]):-
     conc(T1,L,T2).
reverse([],[]).
reverse([H|T],R):-
     reverse(T,R1),
     conc(R1,[H],R).
```

OUTPUT

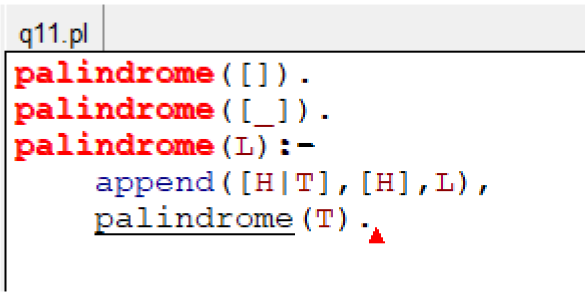


```
?- reverse([2,4],R).
R = [4, 2].

?- reverse([],R).
R = [].

?- reverse([2,3,5,4],R).
R = [4, 5, 3, 2].
```

Q11. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.

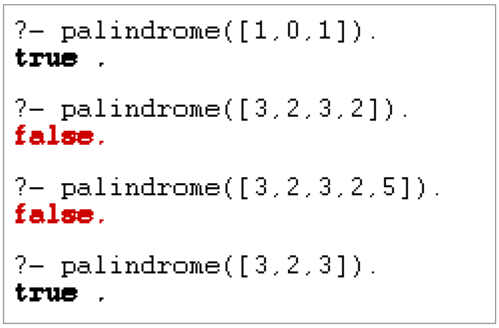
CODE

```
palindrome([]).

palindrome([_]).

palindrome(L):-

   append([H|T],[H],L),

   palindrome(T).
```



```
palindrome([]).
palindrome([_]).
palindrome(L):-
     append([H|T],[H],L),
     palindrome(T).
```

OUTPUT

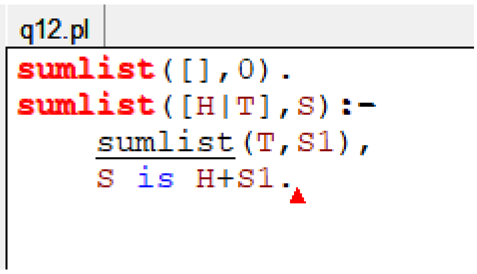


```
?- palindrome([1,0,1]).
true .

?- palindrome([3,2,3,2]).
false.

?- palindrome([3,2,3,2,5]).
false.

?- palindrome([3,2,3]).
true .
```

Q12. Write a Prolog program to implement sumlist(L, S) so that S is the sum of a given list L.

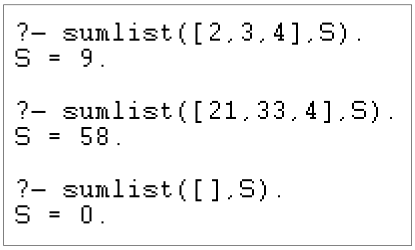
CODE

```
sumlist([],0).

sumlist([H|T],S):-

   sumlist(T,S1),

   S is H+S1.
```



```
sumlist([],0).
sumlist([H|T],S):-
    sumlist(T,S1),
    S is H+S1.
```

OUTPUT

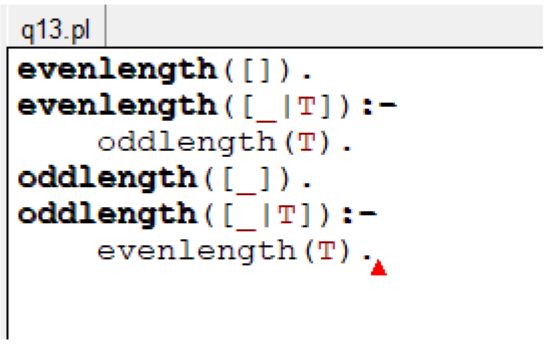


```
?- sumlist([2,3,4],S).
S = 9.

?- sumlist([21,33,4],S).
S = 58.

?- sumlist([],S).
S = 0.
```

Q13. 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.

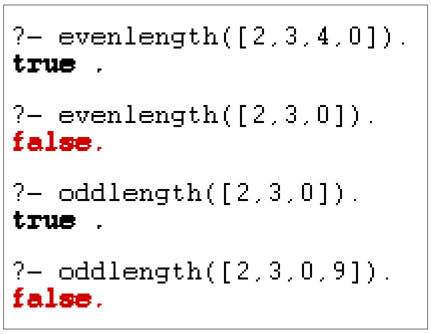
CODE

```
evenlength([]).

evenlength([_|T]):-

   oddlength(T).

oddlength([_]).

oddlength([_|T]):-

   evenlength(T).
```



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

OUTPUT

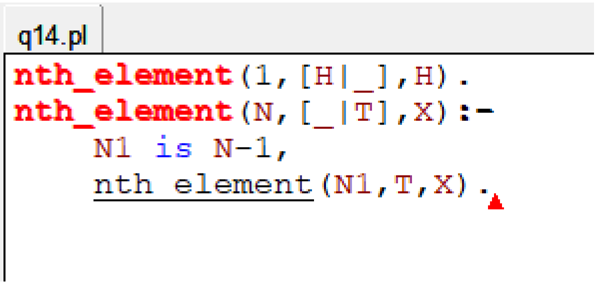


```
?- evenlength([2,3,4,0]).
true .

?- evenlength([2,3,0]).
false.

?- oddlength([2,3,0]).
true .

?- oddlength([2,3,0,9]).
false.
```

Q14. 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.

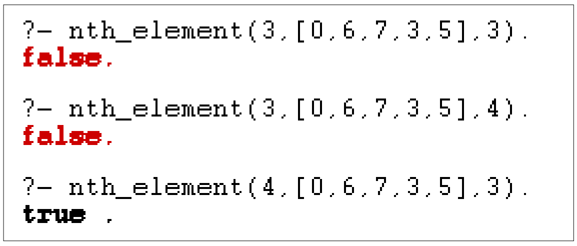
CODE

```
nth_element(1,[H|_],H).

nth_element(N,[_|T],X):-

   N1 is N-1,

   nth_element(N1,T,X).
```



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

OUTPUT

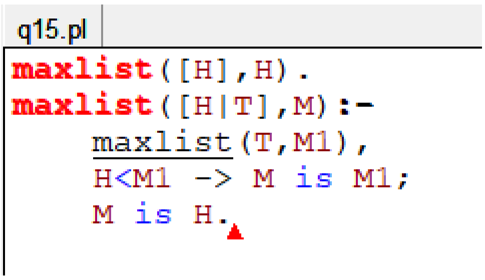


```
?- nth_element(3,[0,6,7,3,5],3).
false.

?- nth_element(3,[0,6,7,3,5],4).
false.

?- nth_element(4,[0,6,7,3,5],3).
true .
```

Q15. Write a Prolog program to implement maxlist(L, M) so that M is the maximum number in the list.

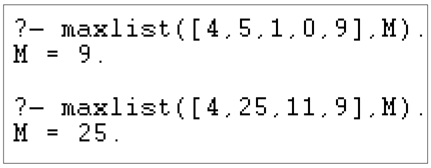
CODE

```
maxlist([H],H).

maxlist([H|T],M):-

   maxlist(T,M1),

   H<M1 -> M is M1;

   M is H.
```



```
maxlist([H],H).
maxlist([H|T],M):-
    maxlist(T,M1),
    H<M1 -> M is M1;
    M is H.
```

OUTPUT

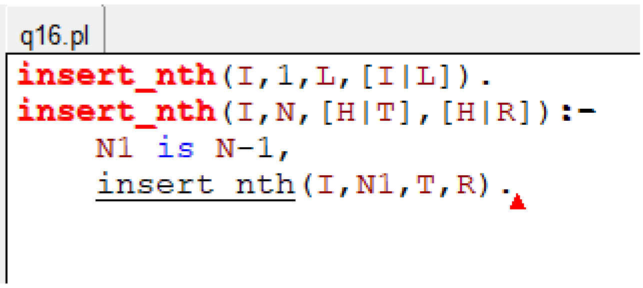


```
?- maxlist([4,5,1,0,9],M).
M = 9.

?- maxlist([4,25,11,9],M).
M = 25.
```

Q16. 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.

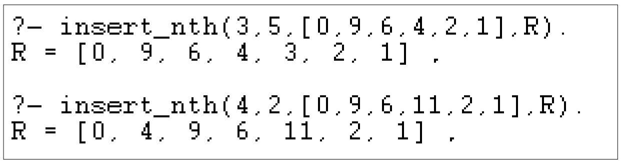
CODE

```
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).
```



```
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).
```

OUTPUT

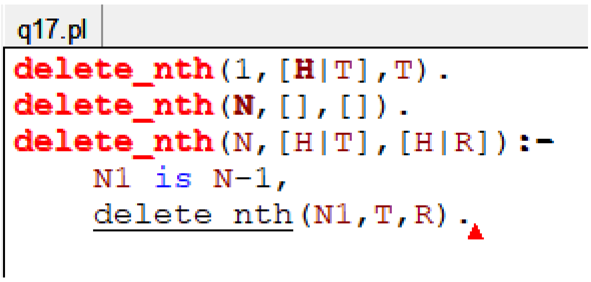


```
?- insert_nth(3,5,[0,9,6,4,2,1],R).
R = [0, 9, 6, 4, 3, 2, 1] .

?- insert_nth(4,2,[0,9,6,11,2,1],R).
R = [0, 4, 9, 6, 11, 2, 1] .
```

Q17. 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.

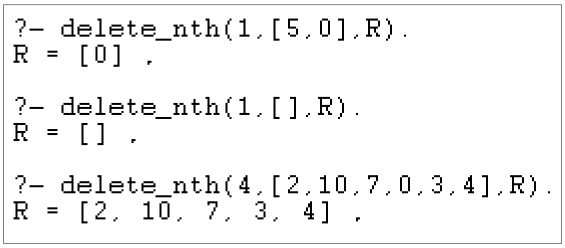
CODE

```
delete_nth(1,[H|T],T).

delete_nth(N,[],[]).

delete_nth(N,[H|T],[H|R]):-

   N1 is N-1,

   delete_nth(N1,T,R).
```



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

OUTPUT

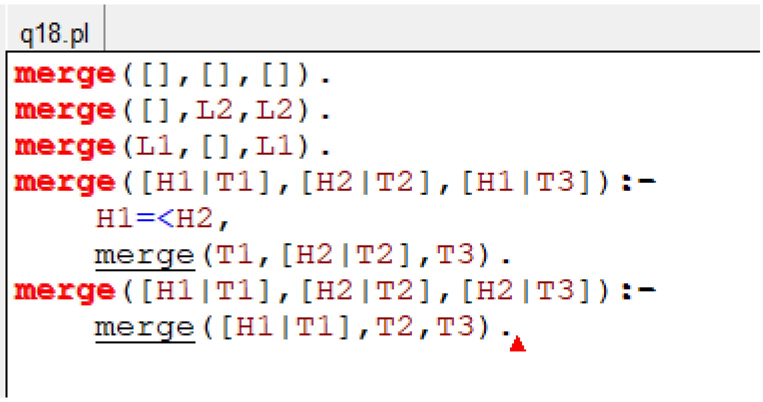


```
?- delete_nth(1,[5,0],R).
R = [0] .

?- delete_nth(1,[],R).
R = [] .

?- delete_nth(4,[2,10,7,0,3,4],R).
R = [2, 10, 7, 3, 4] .
```

Q18. 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.

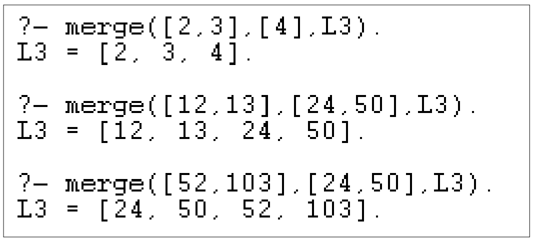
CODE

```
merge([],[],[]).

merge([],L2,L2).

merge(L1,[],L1).

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

   H1=<H2,

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

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

   merge([H1|T1],T2,T3).
```



```
merge([],[],[]).
merge([],L2,L2).
merge(L1,[],L1).
merge([H1|T1],[H2|T2],[H1|T3]):-
      H1=<H2,
      merge(T1,[H2|T2],T3).
merge([H1|T1],[H2|T2],[H2|T3]):-
      merge([H1|T1],T2,T3).
```

OUTPUT



```
?- merge([2,3],[4],L3).
L3 = [2, 3, 4].

?- merge([12,13],[24,50],L3).
L3 = [12, 13, 24, 50].

?- merge([52,103],[24,50],L3).
L3 = [24, 50, 52, 103].
```