% 1. Write a Prolog program to calculate the sum of two numbers.

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

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

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

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

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

factorial(0, 1) :- !.

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

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

fibonacci(0, 0) :- !.

fibonacci(1, 1) :- !.

fibonacci(N, T) :- N > 1, N1 is N - 1, fibonacci(N1, T1), N2 is N - 2, fibonacci(N2, T2), T is T1 + T2.

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

gcd(X, 0, X) :- !.

gcd(X, Y, GCD) :- Y > 0, Z is X mod Y, gcd(Y, Z, GCD).

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

power(\_, 0, 1) :- !.

power(Num, Pow, Ans) :- Pow > 0, Pow1 is Pow - 1, power(Num, Pow1, Ans1), Ans is Num \* Ans1.

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

multi(\_, 0, 0) :- !.

multi(N1, N2, R) :- N2 > 0, N3 is N2 - 1, multi(N1, N3, R1), R is R1 + N1.

% 8. 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).

% 9. 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.

conc([], L2, L2) :- !.

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

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

reverse([], []) :- !.

reverse([H|T], R) :- reverse(T, R1), conc(R1, [H], R).

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

palindrome([]) :- !.

palindrome([\_]) :- !.

palindrome([H

% 11. palindrome

palindrome([]).

palindrome([\_]) :- !.

palindrome([H|T]) :- append(M, [H], T), palindrome(M).

% 12. sumlist

sumlist([], 0).

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

% 13. evenlength and oddlength

evenlength([]).

evenlength([\_,\_|T]) :- evenlength(T).

oddlength([\_]).

oddlength([\_,\_|T]) :- oddlength(T).

% 14. nth\_element

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

nth\_element(N, [\_|T], X) :- N > 1, N1 is N - 1, nth\_element(N1, T, X).

% 15. maxlist

maxlist([X], X) :- !.

maxlist([H|T], M) :- maxlist(T, MT), (H > MT, M = H ; H =< MT, M = MT).

% 16. insert\_nth

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).

% 17. delete\_nth

delete\_nth(1, [\_|T], T) :- !.

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

% 18. merge

merge([], L, L).

merge(L, [], L).

merge([H1|T1], [H2|T2], [H1|R]) :- H1 =< H2, !, merge(T1, [H2|T2], R).

merge([H1|T1], [H2|T2], [H2|R]) :- H1 > H2, !, merge([H1|T1], T2, R).