ARTIFICIAL INTELLIGENCE PRACTICALS

NAME - RITIK YADAV

ROLL NUMBER - 205727

COURSE - B.Sc. (H) COMPUTER SCIENCE

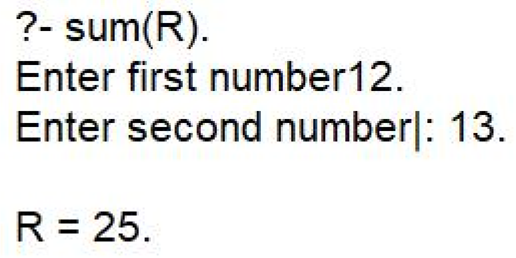
SECTION - B

SEMESTER - VI

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

```
sum(R) :- write("Enter first number"),read(X),write("Enter second number"),read(Y),R is X+Y.
```

OUTPUT

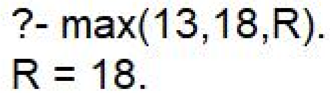


```
?- sum(R).
Enter first number12.
Enter second number|: 13.

R = 25.
```

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,R):-X>Y -> R is X ; R is Y.
```

OUTPUT

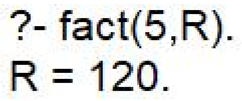


```
?- max(13,18,R).
R = 18.
```

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

```
fact(0,1):-!.
fact(N,R):- N>0,N1 is N-1,fact(N1,R1),R is N*R1.
```

OUTPUT

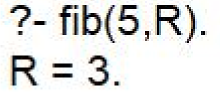


```
?- fact(5,R).
R = 120.
```

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

```
fib(1,0):-!.
fib(2,1):-!.
fib(N,R):- N>0,N1 is N-1,N2 is N-2,fib(N1,R1),fib(N2,R2),R
is R1+R2.
```

OUTPUT

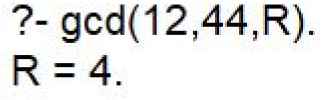


```
?- fib(5,R).
R = 3.
```

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

```
gcd(0,N2,N2):- !.
gcd(N1,0,N1):- !.
gcd(N1,N2,R):- N1<N2, Y is N2-N1,!, gcd(N2,Y,R); Y is
N1-N2 ,!, gcd(N2,Y,R).
```

OUTPUT

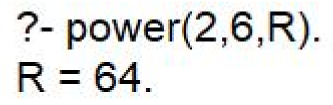


```
?- gcd(12,44,R).
R = 4.
```

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

```
power(N,0,1):- !.
power(N,P,R):- P>0,P1 is P-1,!,power(N,P1,R1),R is R1*N.
```

OUTPUT

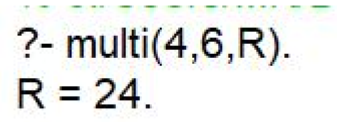


```
?- power(2,6,R).
R = 64.
```

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

OUTPUT

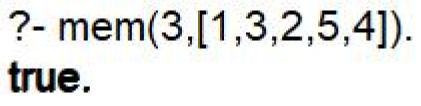


```
?- multi(4,6,R).
R = 24.
```

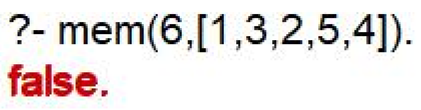
8. Write a Prolog program to implement memb(X, L): to check whether X is a member of L or not.

```
mem(X,[X|Tail]):-!.
mem(X,[H|T]):-mem(X,T).
```

OUTPUT



```
?- mem(3,[1,3,2,5,4]).
true.
```

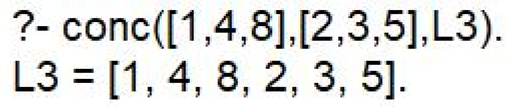


```
?- mem(6,[1,3,2,5,4]).
false.
```

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([],L,L):-!.
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).
```

OUTPUT

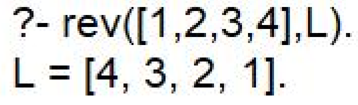


```
?- conc([1,4,8],[2,3,5],L3).
L3 = [1, 4, 8, 2, 3, 5].
```

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

```
conc([],L,L).
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).
rev([],[]).
rev([H|T],Reverse):-rev(T,L1),conc(L1,[H],Reverse).
```

OUTPUT

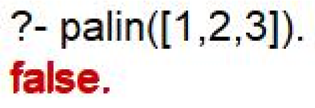


```
?- rev([1,2,3,4],L).
L = [4, 3, 2, 1].
```

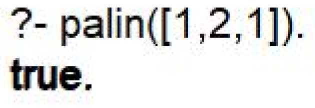
11. Write a program in PROLOG to implement palindrome (L) which checks whether a list L is a palindrome or not.

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

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

palin(L) :- rev(L,L).
```

OUTPUT



```
?- palin([1,2,3]).
false.
```

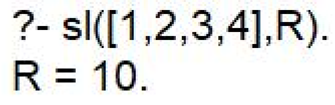


```
?- palin([1,2,1]).
true.
```

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

```
sl([],0).
sl([H|T],R):-sl(T,R1),R is R1+H.
```

OUTPUT

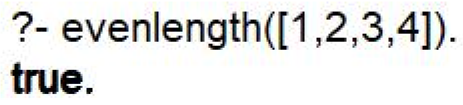


```
?- sl([1,2,3,4],R).
R = 10.
```

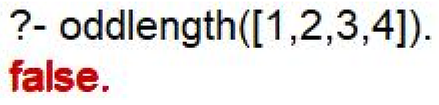
12. Write a Prolog program to implement two predicates evenlength(List) and oddlength(List) so that they are true if their argument is a list of even or odd length respectively.

```
len([],0).
len([_|T],R):- len(T,F1), R is F1+1.
evenlength(L):-len(L,T2),T3 is mod(T2,2),T3==0.
oddlength(L):-len(L,T2),T3 is mod(T2,2),T3==1.
```

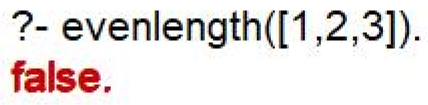
OUTPUT



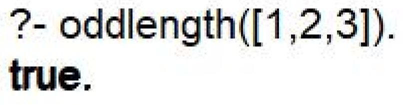
```
?- evenlength([1,2,3,4]).
true.
```



```
?- oddlength([1,2,3,4]).
false.
```



```
?- evenlength([1,2,3]).
false.
```

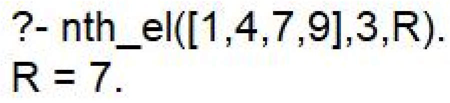


```
?- oddlength([1,2,3]).
true.
```

14. 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_el([],N,[]).
nth_el([H|T],N,R):-N1 is N-1,N1>0,!,nth_el(T,N1,R);R is H.
```

OUTPUT

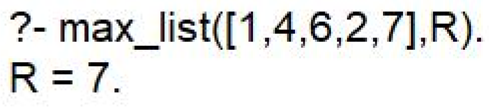


```
?- nth_el([1,4,7,9],3,R).
R = 7.
```

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

```
max_list([X], X):-!.
max_list([X|T], Max) :-max_list(T, Temp),(X > Temp ->
Max = X ; Max = Temp).
```

OUTPUT

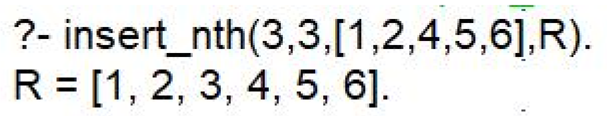


```
?- max_list([1,4,6,2,7],R).
R = 7.
```

16. 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.

```
conc([],L,L).
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).
insert_nth(I,P,[X|Y],[X|M]):- P>1,P1 is
P-1,!,insert_nth(I,P1,Y,M).
insert_nth(I,P,[X|Y],M):- conc([I],[X|Y],M).
```

OUTPUT

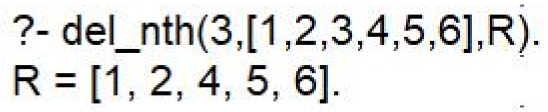


```
?- insert_nth(3,3,[1,2,4,5,6],R).
R = [1, 2, 3, 4, 5, 6].
```

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.

```
conc([],L,L).
conc([X|L1],L2,[X|L3]):-conc(L1,L2,L3).
del_nth(P,[X|Y],[X|M]):- P>1,P1 is P-1,!,del_nth(P1,Y,M).
del_nth(P,[X|Y],M):- conc([],Y,M).
```

OUTPUT

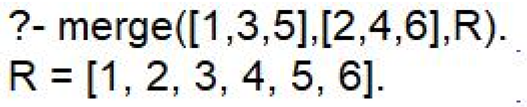


```
?- del_nth(3,[1,2,3,4,5,6],R).
R = [1, 2, 4, 5, 6].
```

18. 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.

```
merge(X,[],X).
merge([],Y,Y).
merge([X|A],[Y|B],[X|C]):-X<Y,!,merge(A,[Y|B],C).
merge([X|A],[Y|B],[X,Y|C]):-X=Y,!,merge(A,B,C).
merge([X|A],[Y|B],[Y|C]):-Y<X,!,merge([X|A],B,C).
```

OUTPUT



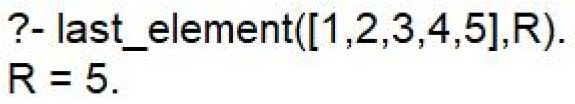
```
?- merge([1,3,5],[2,4,6],R).
R = [1, 2, 3, 4, 5, 6].
```

EXTRA PRATICALS

1. Write a prolog program to implement last_element(L,X) where L is a list and X represents last element of the given list.

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

OUTPUT

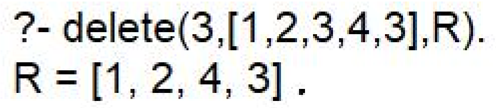


```
?- last_element([1,2,3,4,5],R).
R = 5.
```

2. Write a prolog program to delete first element with predicate.

```
delete(Head, [Head|Tail], Tail).
delete(Item, [Head|Tail], [Head|New_Tail]) :- delete(Item,
Tail, New_Tail).
```

OUTPUT

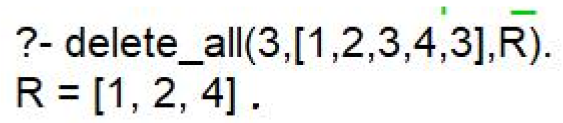


```
?- delete(3,[1,2,3,4,3],R).
R = [1, 2, 4, 3] .
```

3. Write a prolog program to implement delete_all(X,L,R) to delete all occurrences of an element.

```
delete_all(_Head, [], []).
delete_all(Item, [Head|Tail], [Head|New_Tail]) :- Item =\=
Head,delete_all(Item, Tail, New_Tail).
delete_all(Item, [Item|Tail], New_Tail) :- delete_all(Item,
Tail, New_Tail).
```

OUTPUT



```
?- delete_all(3,[1,2,3,4,3],R).
R = [1, 2, 4] .
```

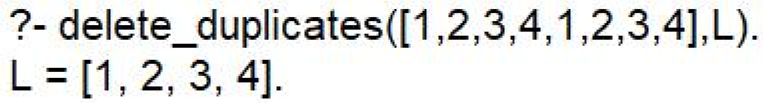
4. Write a prolog program to implement a family tree.

```
parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).
male(tom).
male(bob).
male(jim).
female(liz).
female(pat).
female(ann).
female(pam).
grandparent(X,Z) :- parent(X,Y),parent(Y,Z).
father(X,Y) :- male(X),parent(X,Y).
mother(X,Y) :- female(X),parent(X,Y).
brother(X,Y) :- male(X),parent(Z,X),parent(Z,Y).
sister(X,Y) :- female(X),parent(Z,X),parent(Z,Y).
uncle(X,Y) :- male(X),parent(Z,Y),brother(Z,X).
married(X,Y) :- father(X,Z),mother(Y,Z).
husband(X,Y) :- male(X),couple(X,Y).
wife(X,Y) :- female(X),couple(Y,X).
offspring(X,Y) :- parent(Y,X).
predecessor(X,Y) :- parent(Z,Y);parent(X,Z).
```

5. Write a prolog program to delete all duplicate elements in list.

```
mem(X,[X|Tail]):-!.
mem(X,[H|T]):-mem(X,T).
delete_duplicates([], []).
delete_duplicates([H|T],L) :-mem(H,
T),!,delete_duplicates(T, L).
delete_duplicates([H|T],[H|L]) :-not(mem(H,T)),!,delete_du
plicates(T,L).
```

OUTPUT

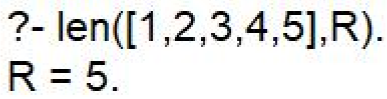


```
?- delete_duplicates([1,2,3,4,1,2,3,4],L).
L = [1, 2, 3, 4].
```

6. Write a prolog program to find the number of element in a list.

```
len([],0).
len([_|T],R):- length(T,F1), R is F1+1.
```

OUTPUT



```
?- len([1,2,3,4,5],R).
R = 5.
```