Esercizio 1

Si definisca un predicato in PROLOG chiamato `maxlist` che applicato ad una lista di liste di interi `ListListInt` dia come risultato la lista degli elementi massimi di ogni lista componente di `ListListInt`. Si definisca prima la versione ricorsiva e poi quella ricorsiva-tail.
Esempio:

```
?- maxlist([[3,10,2], [6,9],[1,2]], X).
    yes, X = [10,9,2]
```

Soluzione:

```
maxlist([],[]):-!.
maxlist([X|Y],[N|T]):-
max(X,N),maxlist(Y,T).
```

Versione ricorsiva

```
max([X],X):-!.
max([X|T],X):- max(T,N),X>=N,!.
max([X|T],N):- max(T,N).
```

Versione iterativa

```
max([X|T],M):- max(T,X,M).
max([],M,M):-!.
max([H|T],MT,M):- H>MT, !,max(T,H,M).
max([H|T],MT,M):- max(T,MT,M).
```

Esercizio 2

Data una lista L1 e un numero intero N, scrivere un predicato Prolog `domanda1(L1,N,L2)` che restituisca in L2 la lista degli elementi di L1 che sono liste contenenti solo due valori interi positivi fra 1 e 9 la cui somma valga N.
Esempio:

```
:- domanda1([[3,1],5,[2,1,1],[3],[1,1,1],a,
[2,2]],4,L2).
    yes, L2 = [[3,1], [2,2]]
```

Soluzione:

```
domanda1([],_,[]).
domanda1([[A,B]|R ], N, [[A,B]|S]):-
    A>=1, A=<9, B>=1, B=<9,
    N is A + B, !,
    domanda1( R,N,S ).
domanda1([_|R], N,S ):- domanda1( R,N,S ).
```

Esercizio 3

Si definisca un predicato in PROLOG chiamato `averStud` che applicato a un numero di matricola di uno studente `Matr` e a una lista di esami `LE` dia come risultato la media `AV` dei suoi voti. Ogni esame sia rappresentato da un termine della lista `LE` della forma `esame(Matr,Esame,Voto)`. Si definisca prima la versione ricorsiva e poi quella ricorsiva-tail.

Esempio:

```
?-averStud(s1,[esame(s2,f1,30),
esame(s1,f1,27),esame(s3,f1,25),
esame(s1,f2,30)], AV).
   yes, AV = 28.5
```

Soluzione:

% versione ricorsiva

```
averStud(S,L,AV) :-
    totStud(S,L,N,T),
    N > 0,
    AV is T/N.
totStud(_,[],0,0) :- !.
totStud(S,[esame(S,_,V)|R],N,T) :- !,
    totStud(S,R,NN,TT),
    N is NN + 1, T is TT + V.
totStud(S,[_|R],N,T) :-
    totStud(S,R,N,T).
```

% versione tail-ricorsiva

```
averStud(S,L,AV) :-
    totStud(S,L,0,N,0,T),
    N > 0,
    AV is T/N.
totStud(_,[],N,N,T,T) :- !.
totStud(S,[esame(S,_,V)|R],NI,NO,TI,TO):- !,
    N is NI + 1, T is TI + V,
    totStud(S,R,N,NO,T,TO).
totStud(S,[_|R],NI,NO,TI,TO) :-
    totStud(S,R,NI,NO,TI,TO).
```

Esercizio 4

Un associazione di volontariato tiene traccia delle spese di cancelleria tramite un insieme di fatti Prolog del tipo:

```
spesa(carta, 13).
spesa(matite, 12).
spesa(graffette, 5.99).
spesa(carta, 12).
```

Dove il primo parametro rappresenta l'oggetto acquistato, ed il secondo l'importo.

Definire i seguenti predicati Prolog:

1. elenca(L), che deve restituire la lista degli item acquistati, senza ripetizione. Si usi a tal scopo il predicato setof.
2. subTotale(X, S), che ricevuto in ingresso un oggetto X, restituisce l'ammontare della spesa per tale oggetto
3. totale(S), che restituisce quanto è stato speso in cancelleria.

Ad esempio, i predicati (invocati sulla base di conoscenza presentata all'inizio di questo esercizio) devono restituire:

```
| ?- elenca(L).
L = [carta,graffette,matite] ?

| ?- subtotale(carta, S).
S = 25 ?

| ?- totale(S).
S = 42.99 ?
```

Soluzione:

Se si usa il predicato setof è scorretto a meno di utilizzare la quantificazione sulla variabile Y.

```
    elenca(L) :-
        setof( X, Y^spesa(X, Y), L).
Oppure :

    elenca1(L1) :-
        findall( X, spesa(X, _), L),
        eliminarip(L,L1).


eliminarip([],[]):-!.
eliminarip([A|Rest1], Rest3):-
  member(A,Rest1),!,eliminarip(Rest1,Rest3).
eliminarip([A|Rest1], [A|Rest3]):-
    eliminarip(Rest1,Rest3).


    subtotale(X, S) :-
        bagof(V, spesa(X, V), L),
        somma(L, S).

    totale(S) :-
        findall(V, spesa(_, V), L),
        somma(L, S).

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

Esercizio 5

Si scriva un programma Prolog che, prendendo in ingresso due liste L1 e L2, restituisca in uscita due liste L3 e L4 tali che L3 contenga gli elementi di L1 che appartengono anche a L2, mentre L4 contenga gli elementi di L1 che non appartengono a L2. Si supponga disponibile il predicato member. Si dica inoltre se il predicato così definito è ricorsivo tail.
Esempio:
`?-list_mem([a,r,t],[t,s,m,n,a],L3,L4).`
restituirà `L3=[a,t]` e `L4=[r].`

Soluzione:

```
list_m([],L2,[],[]).
list_m([A|Rest1],L2,[A|Rest3],L4):-
    member(A,L2),!,list_m(Rest1,L2,Rest3,L4).
list_m([A|Rest1],L2,L3,[A|Rest4]):-
    list_m(Rest1,L2,L3,Rest4).
```

Il predicato è ricorsivo tail.

Esercizio 6

Si scriva un predicato Prolog che data una lista ed un elemento `El` appartenente alla lista, restituisca in uscita l'elemento successivo ad `El` nella lista.
Esempio:

```
?- consec(3, [1,7,3,9,11],X).
yes   X=9
```

Nel caso in cui `El` sia l'ultimo elemento il predicato dovrà fallire.

Soluzione

```
consec(El, [El| [X|_]],X):-!.
consec(El, [_|Tail],X):- consec(El,Tail,X).
```

Esercizio 7

Si scriva un predicato Prolog `list_to_set` a due argomenti che data una lista di liste come primo argomento leghi il secondo argomento a una lista nella quale sono state eliminate le liste ripetute o le loro permutazioni.
Per esempio dato il goal:

```
?-list_to_set([[1,2,3],[3,1,2],[1]], Y).
```

si vuole ottenere:

```
yes   Y=[[3,1,2],[1]]
```

Per esempio dato il goal:

```
?-list_to_set([[1,2],[1,2],[1,2]], Y).
```

si vuole ottenere:

```
yes   Y=[[1,2]]
```

Si supponga dato il predicato `permutation(X,Y)` che verifica se una lista `X` è una permutazione della lista `Y`.

Soluzione

```
list_to_set([],[]):- !.
list_to_set([I|R],[I|R1]):-
   not member list(I,R),!,
   list_to_set(R,R1).
list_to_set([I|R],R1):- list_to_set(R,R1).

member_list(X,[Y|_]):-permutation(X,Y),!.
member_list(X,[_|R]):-member_list(X,R).
```

Esercizio 8

Si scriva un programma Prolog che data in ingresso una lista di liste con 2 elementi ciascuna ed una costante c1 restituisca in uscita due liste DX ed SX, la prima contenente gli elementi che nelle coppie compaiono a destra di c1, la seconda a sinistra.

Soluzione

```
coppie([],_,[],[]).
coppie([[X,X]|T],X,[X|Td],[X|Ts]) :- !,
   coppie(T,X,Td,Ts).
coppie([[X,Y]|T],X,[Y|Td],Ts) :- !,
   coppie(T,X,Td,Ts).
coppie([[Y,X]|T],X,Td,[Y|Ts]) :- !,
   coppie(T,X,Td,Ts).
coppie([_|T],X,Td,Ts) :-
   coppie(T,X,Td,Ts).
```

Esercizio 9

Scrivere un predicato Prolog che fornisce l'ultimo elemento di una lista.

Esercizio 10

Si definisca un predicato Prolog che calcola i massimi locali (esclusi gli estremi) di una lista, ad esempio:
:- max_loc([5,4,7,2,3,6,1,2],X)
yes, X=[7,6]

Esercizio 11

Dare un programma in Prolog che definisca la relazione tra due liste di avere l'una lunghezza doppia dell'altra.

Esercizio 12

Scrivere un predicato Prolog per verificare se una lista è palindroma

Esercizio 13

Dato un albero binario, si scriva un predicato che calcola la profondità massima dell'albero.

Esercizio 14

Dare un programma in Prolog che definisca la relazione nodiInterni tra un albero binario e un naturale, tale che il numero naturale indichi il numero di nodi interni (non foglie) dell'albero

Esercizio 15

Dati due alberi A1 ed A2 si scriva un predicato che verifica se A1 è un sottoalbero di A2.

Esercizio 16

Data una matrice *N*x*N* rappresentata come lista di liste, si calcoli la somma degli elementi della diagonale principale.

Esercizio 17

Una matrice M di bit $n \times m$, si dice matrice C-Grey se, $\forall j = 1,\ldots,m-1$, la j-esima colonna della matrice differisce dalla colonna $(j+1)$-esima esattamente in un bit. Si realizzi un programma Prolog che, data M rappresentata come lista di colonne, risponda sì se M è C-Grey.

Esercizio 18

Scrivere un predicato `flatten` che "appiattisce" una lista di liste. Ad esempio:

```
:- flatten([1,a,[2,3],[],h,f(3),[c,[d,[e]]]],L).
yes, L=[1,a,2,3,h,f(3),c,d,e]
```