**Definire una funzione removeElementsByValues che prende in input due liste di interi: values e filters e  restituisce una nuova lista contenente solo gli elementi di values che non appaiono nella lista filters  nell'ordine in cui appaiono nella lista values.**

**removeElementsByValues :: [Int] -> [Int] -> [Int]**

**Esempio:**

**removeElementsByIndices [1,4,2,1] [2,1] restituisce [4]**

removeElementsByValues:: [Int] -> [Int] -> [Int]

removeElements [] \_ = []

removeElements (v: vs) fs | not (elem v fs) = v: removeElements vs fs

| otherwise = removeElements vs fs

elem:: Int -> [Int] -> Bool

elem \_ [] = False

elem n (x:xs) | x==n = True

| otherwise = elem n xs

**Scrivere una funzione sumOfLists che riceve in input una lista di liste di numeri l e restituisce una lista di numeri l1 tale che ogni elemento in posizione i-esima di l1 corrisponda alla somma degli elementi nella lista in posisione i-esima di l**

**sumoflists :: Num a => [[a]] -> [a]**

sumOfLists:: Num a => [[a]] -> [a]

sumOfLists [] = []

sumOfLists (x:xs) = sum x : sumOfLists xs

sum:: Num a => [a] -> a

sum [] = 0

sum (x:xs) = x + sum xs

**Qual'è il tipo dell'espressione seguente?**

**map (+1)**

**Motivare la risposta.**

map (+1):: Num a => [a] -> [a]

la funzione map riceve come argomenti, una funzione f e una lista l, restituendo la lista l a cui ad ogni elemento viene applicata la funzione f. Siccome in questo caso la funzione f è (+1), la lista deve necessariamente essere una lista di numeri

**Definire un predicato prolog subsequences/2 che abbia come primo e secondo termine due liste L1 e L2  tali che**

**subsequences(L1, L2) restisca vero se e solo se L2 si può ottenere da L1 rimuovendo alcuni dei suoi elementi (anche nessuno o tutti)**

**esempi:**

* **subsequences([1,2,3],[2,1]) è falso**
* **subsequences([1,2,3],[1,3]) è vero**
* **subsequences([1,2,3],[1,5]) è falso**
* **subsequences([1,2,3],[]) è vero**

**il predicato deve funzionare (almeno) in modalità +, +, ma è preferibile che funzioni anche in modalità +,-**

subsequences(L1,L1).

subsequences(L1,[]).

subsequences([T1|C1], [T1|C2]):- subsequences(C1,C2).

subsequences([T1|C1], [T2|C2]:- subsequences(C1, [\_|C2]).

**Supponendo di avere una base di fatti per i seguenti predicati: persona/2, amico/2, maschio/1, femmina/1, dove persona contenga come primo termine il nome della persona e come secondo la sua età**

**Si definiscano i predicati seguenti:**

1. **soloamicifemmine(X): vero se e solo X ha solamente amici di sesso femminile (nel caso X non abbia amici il predicato è vero)**
2. **amicistessaeta(X): vero se e solo tutti almeno una coppia di amici di X hanno la medesima età**
3. **stessiamici(X,Y): vero se e solo se X e Y hanno degli insiemi di amici AX e AY tali che AX e AY coincidono**

amico(X,Y):-amico(Y,X).

soloamicifemmine(X):- persona(X,\_),

findall(AMICO, amico(X,AMICO), AMICI),

verificaf(AMICI).

verificaf([]).

verificaf([T|C]):- femmina(T), verifica(C).

amicistessaeta(X):- persona(X,\_),

amico(X,AMICO1),

amico(X,AMICO2),

persona(AMICO1, ETA), persona(AMICO2,ETA).

stessiamici(X,Y):- persona(X), persona(Y),

findall(AMICO, amico(X,AMICO), AX),

findall(AMICO, amico(Y, AMICO), AY).

coincidono(AX,AY), coincidono(AY,AX).

coincidono([T|C],AY):- elem(T,AY), coincidono(C,AY).

elem(E, [E]).

elem(E, [E|\_]).

elem(E, [\_|C]):- elem(E,C).