**Задача 3.** *Задачата да се реши на един от езиците Scheme или Haskell. По-долу оградете името на езика, който сте избрали за решението си.*

"Етикет" наричаме наредена двойка от низове — име и стойност, а "анотирана данна" наричаме наредена двойка от данна и списък от етикети за нея. Разглеждаме база данни, представена като списък от анотирани низове. "Анотатор" наричаме функция, която приема данна (низ) и връща списък от етикети за нея. Да се попълнят по подходящ начин празните полета по-долу, така че при подаване на база данни `db` и списък от анотатори `annotators`, функцията `annotate` да връща актуализирана база данни, в която за всяка данна в `db` са добавени етикетите, върнати за нея от анотаторите в `annotators`. Ако даден етикет вече съществува за дадена данна, той да не се добавя повторно. Да се реализира помощната функция `addIfNew`, така че да добавя елемента в `x` в началото на списъка `l` само ако `x` не се среща вече в `l`.

***Упътване:*** *могат да се използват наготово функциите `append`, `apply`, `concat`, `concatMap`, `elem`, `filter`, `foldr`, `map`, `member`, `sum` и стандартните функции в R5RS за Scheme и в Prelude за Haskell.*

## Scheme

```
(define (addIfNew x l) ____________________________________________)
(define (annotate db annotators)
  (____________
    (lambda (item-labels-pair)
      (let ((item (car item-labels-pair)) (labels (cdr item-labels-pair)))
        (cons item (_____________ addIfNew labels
                     (_________________________________
                       (lambda (annotator) ____________________________________) annotators))))) db))
```

***Пример:***

```
(define db (list (cons "scheme" (list (cons "typing" "dynamic") (cons "evaluation" "strict")))
                 (cons "haskell" (list (cons "typing" "static"))) (cons "c++" (list))))
(define (evaluation lang)
  (case lang (("scheme")  (list (cons "evaluation" "strict") (cons "macros" "true")))
             (("haskell") (list (cons "evaluation" "lazy"))) (("c++") (evaluation "scheme"))))
(define (purity lang) (if (eqv? lang "haskell") (list (cons "pure" "true")) (list)))
(annotate db (list evaluation purity)) ⟶
  (("scheme" ("macros" . "true") ("typing" . "dynamic") ("evaluation" . "strict"))
   ("haskell" ("evaluation" . "lazy") ("pure" . "true") ("typing" . "static"))
   ("c++" ("evaluation" . "strict") ("macros" . "true")))
```

## Haskell

```
addIfNew x l = ____________________________________________
annotate db annotators =

  _____________
    (\(item, labels) ->
      (item, _____________ addIfNew labels
               (_________________________________
                 (\annotator -> ___________________________________) annotators))) db
```

***Пример:***

```
db = [("scheme", [("typing", "dynamic"), ("evaluation", "strict")]),
      ("haskell", [("typing", "static")]), ("c++", [])]
evaluation "scheme"  = [("evaluation", "strict"), ("macros", "true")]
evaluation "haskell" = [("evaluation", "lazy")]
evaluation "c++"     = evaluation "scheme"
purity lang = if lang == "haskell" then [("pure", "true")] else []
annotate db [evaluation, purity] ⟶
  [ ("scheme",  [ ("macros", "true"), ("typing", "dynamic"), ("evaluation", "strict") ] ),
    ("haskell", [ ("evaluation", "lazy"), ("pure", "true"), ("typing", "static") ] ),
    ("c++", [ ("evaluation", "strict"), ("macros", "true") ] ) ]
```