```haskell
-- Model solution for Tut05
--
-- Copyright [2000..2004] Manuel M T Chakravarty

module Tut05
where

-- the following definitions are given their maximally generic type - even if
-- the tutorial question didn’t ask for it

-- Delete all occurences of the given value from a list
--
-- Example: delete ’x’ "x-files sux" = "-files su"
--
delete                         :: Eq a => a -> [a] -> [a]
delete x []                     = []
delete x (y:ys) | x == y        = delete x ys
                | otherwise     = y : delete x ys

-- Substitute elements in a list
--
-- Example: substitute ’e’ ’i’ "eigenvalue" = "iiginvalui"
--
substitute                          :: Eq a => a -> a -> [a] -> [a]
substitute x y []                    = []
substitute x y (z:zs) | x == z       = y : substitute x y zs
                      | otherwise    = z : substitute x y zs

-- Determine the shortest and the longest string in a list
--
-- Examples: shortestAndLongest ["abc"] = ("abc","abc")
--           shortestAndLongest ["This", "sentence", "is", "ridiculous"] =
--            ("is","ridiculous")
--
shortestAndLongest              :: [String] -> (String, String)
shortestAndLongest []            = error "shortestAndLongest: empty list"
shortestAndLongest [str]         = (str, str)
shortestAndLongest (str:strs)    =
  let
    (shortest, longest) = shortestAndLongest strs
  in
  if length str < length shortest
  then
    (str, longest)
  else if length str > length longest
  then
    (shortest, str)
  else
    (shortest, longest)

-- Yield a list of all the properties associated with all occurrences of a key
-- in an association list
--
-- Examples: lookupAll
--             "Plates"
--             [("Forks", 10), ("Plates", 5), ("Cups", 0), ("Plates", 1)] =
--             [5,1]
--           lookupAll
--             "Knives"
--             [("Forks", 10), ("Plates", 5), ("Cups", 0), ("Plates", 1)] =
--             []
--
lookupAll                                :: Eq a => a -> [(a, b)] -> [b]
lookupAll key []                          = []
lookupAll key ((key’, val) : keyVals)
  | key == key’                           = val : lookupAll key keyVals
  | otherwise                             = lookupAll key keyVals
```