```
(require 2htdp/image)
(require spd/tags)

(@problem 1)
;; An online item selling system (like eBay) allows
sellers to list
;; items to be sold for up to 30 days and track the status
of these items.
;;
;; The status of an item is either:
;; still listed with the number of days left in the
listing, reserved by an
;; interested buyer, or sold.
;;
;; Provide a complete data definition for the status of an
item listing.
```

```
(@problem 2)
;; This problem is broken into four parts. Read all four
parts before you start.
;;
;; You are provided with a signature, purpose, stub, and
check-expects for
;; the function strings-of-len.
;;
;; You must include the function definition
;; for strings-of-len in each of Parts A-D.


(@htdf strings-of-len)
(@signature (listof String) Natural -> (listof String))
;; produce a list of strings from strings in los that have
length greater than n
(check-expect (strings-of-len empty 5) empty)
```

```
(check-expect (strings-of-len (list "ab" "cde" "ghih") 3)
(list "ghih"))
(check-expect (strings-of-len (list "ab" "cde" "ghih") 2)
(list "cde" "ghih"))

(define (strings-of-len los n) empty)


;; Problem 2 (PART A)
;;
;; Write a non-tail recursive function definition for
strings-of-len.
;; Do not use function composition or built-in abstract
functions.
;;
;; A grade of 0 will be given for tail recursive solutions
or
;; solutions that use built-in abstract functions.
```

```
;; Problem 2 (PART B)
;;
;; Write a new function definition using built-in abstract
list functions.
;;
;; A grade of 0 will be given for solutions that do not
use built-in abstract
;; list functions.
```



```
;; Problem 2 (PART C)
;;
;; Write a tail recursive function.
```

```
;; Do not use function composition or built-in abstract
functions.
;;
;; A grade of 0 will be given for solutions that are not
tail recursive,
;; or solutions that use built-in abstract functions.
```



```
;; Problem 2 (PART D)
;;
;; Write a new version of the function definition using
for-each.
;;
;; A grade of 0 will be given for solutions not using
for-each.
```

```
(@problem 3)
;; Design a function that consumes a list of images and
produces a number
;; representing the length of the longest sequence of
images that increase in
;; area.
;;
;; For example:
;;
;;
https://s3.amazonaws.com/edx-course-spdx-kiczales/HTC/probl
ems/17w2flist3.PNG
;;
;; Calling the function with the list in the above link
produces 3, as the first
;; 3 images in a row increase in area. There is another
sequence where images
;; increase in area (the last two images), but the
sequence of 3 is longest of
;; the two sequences in the given list.


(define LOI0 empty)
(define LOI1 (list (square 30 "solid" "blue")
                   (square 40 "solid" "red")
                   (square 70 "solid" "purple")
                   (square 50 "solid" "orange")
                   (square 60 "solid" "pink")))

(define LOI2 (list (square 40 "solid" "yellow")
                   (square 40 "solid" "purple")))

(define LOI3 (list (square 30 "solid" "black")
                   (square 25 "solid" "brown")
                   (square 20 "solid" "grey")))
```

```
(define LOI4 (list (square 50 "solid" "red")
                   (square 20 "solid" "yellow")
                   (square 30 "solid" "purple")))
```

```
;; Problem 4 setup
;;
;; This problem will use words and mutations on words. A
mutation occurs
;; when a letter is added or removed from a word to form a
new word.
;;
;; A given word has a name and a list of mutations that
can be applied to it.
;;
;; A mutation can either add or remove a single letter
from a word to form
;; a new word.
;;
;; The following image illustrates the relationship
between words and mutations:
;;
;;
https://s3.amazonaws.com/edx-course-spdx-kiczales/HTC/probl
ems/17w2fwords.PNG
;;
;; In the image above words are shown in the boxes, and
mutations are shown as
;; arrows. The letter being added or removed in the
mutation to form a new word
;; is shown beside the arrow. The arrow points to the new
word formed.
;;
;; Note: Dashed (- - -) arrows denote a mutation where a
letter is being
;;       removed.
;;       Solid arrows denote a mutation where a letter is
being added.
;;
;; In the English language, there are more relationships
between words and
```

```
;; mutations than shown in the image above.
;; To simplify the problem, use only the examples that are
shown in the image
;; above.
;;
;; A data definition to represent this information has
been provided on the next
;; page.
```



```
;; ================
;; Data Definitions:

(@htdd Word)
(define-struct word (name mutations))
;; Word is (make-word String (listof Mutation))
;; interp. a word with a name composed of lower-case
English letters, and
;;         a list of mutations that can be applied to form
new words


(@htdd Mutation)
(define-struct mut (add? letter result))
;; Mutation is (make-mut Boolean String Word)
;; interp. a mutation on a word that adds or removes a
letter to form a new word
;;         add? is true when the mututation adds a letter,
false for a removal
;;         letter is the letter being added or removed
```

```
;;         result is the new word that is formed
;; ASSUME: letter is always a one-letter lowercase String

(define WORDS-GRAPH
  (shared [(FEAST (make-word "feast" (list (make-mut false
"e" FAST)
                                           (make-mut false
"f" EAST))))
           (BEAST (make-word "beast" (list (make-mut false
"b" EAST))))
           (EAST  (make-word "east"  (list (make-mut true
"f" FEAST)
                                           (make-mut false
"s" EAT))))
           (BEAT  (make-word "beat"  empty))
           (FEAT  (make-word "feat"  (list (make-mut false
"f" EAT))))
           (FAST  (make-word "fast"  (list (make-mut false
"s" FAT))))
           (FAT   (make-word "fat"   (list (make-mut true
"e" FEAT))))
           (EAT   (make-word "eat"   (list (make-mut true
"b" BEAT)
                                           (make-mut false
"e" AT))))
           (AT    (make-word "at"    (list (make-mut true
"f" FAT)
                                           (make-mut true
"e" EAT)
                                           (make-mut true
"b" BAT))))
           (BAT   (make-word "bat"   (list (make-mut true
"e" BEAT)))))]
    (list FEAST BEAST EAST BEAT FEAT FAST FAT EAT AT BAT)))

(define FEAST (first WORDS-GRAPH))
(define BEAST (list-ref WORDS-GRAPH 1))
(define EAST (list-ref WORDS-GRAPH 2))
(define BEAT (list-ref WORDS-GRAPH 3))
```

```
(define EAT (list-ref WORDS-GRAPH 7))
(define BAT (list-ref WORDS-GRAPH 9))

#;
(define (fn-for-word w0)
  (local [(define (fn-for-word w)
            (... (word-name w)
                 (fn-for-lom (word-mutations w))))

          (define (fn-for-lom lom)
            (cond [(empty? lom) (...)]
                  [else
                   (... (fn-for-mut (first lom))
                        (fn-for-lom (rest lom)))]))

          (define (fn-for-mut m)
            (... (mut-add? m)
                 (mut-letter m)
                 (fn-for-word (mut-result m))))]
    (fn-for-word w0)))
```



```
(@problem 4)
;; Design a tail-recursive function that consumes a Word
from the words graph
;; and a single letter.
;;
;; The function should produce true if starting at the
word it is possible
;; to reach more add mutations the letter is part of than
remove mutations.
;;
;; For example, starting at FEAST, the letter "f" is in
the following mutations:
;;  - a removal mutation going from FEAST to EAST
;;  - an add mutation going from EAST to FEAST
;;  - a removal mutation going from FEAT to EAT
```

```
;;  - an add mutation going from AT to FAT
;;
;; The function should produce false since "f" is not in
more add than
;; remove mutations (there are 2 of each) in the graph.
;;
;; On the other hand, the function would produce true if
given the letter "e",
;; as there are 3 add mutations with "e", and only 2
remove mutations.
;;
;; You must provide a tail-recursive function definition.
;; The signature, purpose, and check-expects have been
provided for you.


(@htdf more-add-than-remove?)
(@signature World String -> Boolean)
;; produce true if x is in more add than remove mutations
reachable from w0
;; ASSUME: x is a String that is only one lower-case
letter long
(check-expect (more-add-than-remove? BEAST "e") false)
(check-expect (more-add-than-remove? BAT "e") true)
(check-expect (more-add-than-remove? FEAST "f") false)
(check-expect (more-add-than-remove? FEAST "e") true)
(check-expect (more-add-than-remove? BEAST "b") true)
(check-expect (more-add-than-remove? FEAST "s") false)

(define (more-add-than-remove? w x) true)
```

```
;; You may use this page to continue your answer from
Problem 5
```