;; NOTE:
;; THIS FINAL EXAM WAS FOCUSED PRIMARILY ON THE LATTER PART OF THE COURSE.
;; THAT MADE IT SOMEWHAT HARDER THAN INTENDED. IT IS VERY USEFUL FOR REVIEW,
;; BUT BE SURE TO ALSO LOOK AT OTHER EXAMS THAT INCLUDE SOME EASIER PROBLEMS.

;; ---------------------------------------------------------------------------
;; ---------------------------------------------------------------------------
;; Problem 1.

Write the function definition only (no signature, purpose, or tests) for each of the following functions. In each case you should use one or more built in abstract list functions, possibly with a closure.

(A) A function that consumes a color and a list of number and produces a list of squares of the given color and size. So

(squares "blue" (list 40 60)) produces (list □□□□)
(B) A function that consumes a list of strings and produces the length of the longest string.
Problem 2

In this problem you will design a function to produce a image of concentric outline circles of given radii. For example

\[(\text{circles (list 10 20 60)})\] should produce

Use a for-each loop and a single accumulator.

Also recall that:
- \[(\text{circle n "outline" "black"})\] produces an outline circle with radius n.
- overlay can be used to overlay 2 images
- and that because they are outline circles the order in which they are overlaid does not matter

Your design should include signature, purpose, tests and working function. You may wish to also write out a separate copy of the template you use.
Problem 3
Consider the following type comments:

(define-struct node (key val l r p))
;; BST is one of:
;; - false
;; - (make-node Natural String BST BST (listof Natural))
;; interp. A binary search tree where each node has key, value
;; and left/right children. If the tree has been
;; 'decorated' then p is a list of keys as described
;; below.
;; Note the invariant that for a given node n, every
;; node in its left branch has a smaller key and every
;; node in its right branch has a larger key.

;; For any node in the tree, there is a single path from the root to the node.
;; That path can be represented as a list of the keys of the nodes on the path.
;; For example, in this tree

```
  4:d
 /   \
2:b   6:f
 /     /
1:a   3:c   5:e
   /   /
  7:g
```

the path to node 3 would be represented as (list 4 2 3).

Design a function called decorate that consumes a bst and produces a
 corresponding bst in which each node has been decorated with the path to the
 node. So, for example, in the result tree, the node for 1 would be
 (make-node 1 "a" false false (list 4 2 1)).

You will need to use a single accumulator.

You will find it helpful to use the DrRacket function reverse, which given
 a list produces a list of the same values in reverse order.
Problem 4:

This problem involves a directed acyclic graph. It is a SIMPLIFIED version of the web crawler from class with NO accumulators or tail recursion.

Consider the following type comments for a simple model of the web. (Note that unlike the the version from lecture each page also has a list of strings that represents the text on the page.

;; PageName is String
;; interp. the name of a web page
;; note, get-page is a selector-like function that produces
;; the Page with a given PageName.
;; (YOU MAY SIMPLY ASSUME THIS FUNCTION EXISTS)

(define-struct page (name links text))
;; Page is (make-page PageName (listof PageName) (listof String))
;; A page, with:
;; - an intrinsic name
;; - the names of pages it links
;; - and the text on the page
;;

;; (listof PageName) is an abbreviation for the usual ListOfPageName thing

(define PAGES
  (list (make-page "A" (list "B" "C") (list "dog" "cat" "flower"))
        (make-page "B" (list "D" "E") (list "to" "two" "too"))
        (make-page "C" (list "E") (list "dog" "tree" "dog"))
        (make-page "D" empty empty)
        (make-page "E" (list "F") (list "red" "green"))
        (make-page "F" empty (list "dog")))))

(A)

Do a reference analysis by neatly annotating the type comments above with a line from each type reference to the the corresponding type definition. All your arrows should have the 'pointy-end' at one of the names that appear right before 'is'. Neatly label each line with one of MR, SR or R, depending on whether the reference is a mutual reference, self reference or an ordinary reference.
Write the template for a function operating on PageName. Include any additional template functions for types involved in mutual recursion cycles.
(C)

Design a function to count the number of times a given word appears in the part of the web reachable from a given page. The function should consume PageName and String and produce Natural. As examples:

(count-occurrences "A" "dog") produces 5
(count-occurrences "D" "dog") produces 0
(count-occurrences "F" "dog") produces 1

FOR SIMPLICITY, YOU MAY COUNT A WORD EACH TIME IT IS REACHED, WHICH IS WHY THE FIRST EXAMPLE ABOVE PRODUCES 5.
(EXTRA CREDIT)

Rewrite the count-occurrences function so that it BOTH handles cyclic graphs and is tail recursive. It should also only count each word once, even it is reachable along multiple paths. Since this is for extra credit your solution will need to be almost completely correct to receive points.
(use this page for extra space for earlier problems. be sure to clearly label what problem anything you write is the answer to.)
(use this page for extra space for earlier problems. be sure to clearly label what problem anything you write is the answer to.)