Problem 1a

Given the following partial data definition for a ListOfString:

;; ListOfString is one of:
;; - empty
;; - (cons String ListOfString)

(define LOS1 empty)
(define LOS2 (list "I" "love" "CPSC110"))

Provide the template for a function that operates on an argument of type ListOfString. List the template rules you used in creating the template.
Problem 1b

Now modify your template so that it also expects an additional atomic parameter. You don't have to write down any additional template rules.

Problem 1c

Using your template from Problem 1b, provide the full design of a function called 'count-matches' which consumes a target String and a ListOfString and produces a Natural. Your function should produce the number of strings contained in ListOfString that match the target string. You don't have to provide a stub or rewrite your template.
Problem 2

Design a function named 'sum-of-squares' which consumes (listof Number) and produces Number. The function should produce the sum of the squares of all the numbers in the list. Use an accumulator and tail recursion in your design. You need not provide the stub or template, but you must provide all the other design elements. (Remember that adding an accumulator requires comments about the accumulator.)

It may help to first design 'sum-of-squares' without an accumulator and tail recursion. You’re not required to do so, but if you do, please do that design work on one of the blank pages at the end of the exam, not below. Adding extra clutter below will only confuse your markers, which could result in fewer marks for your efforts.
Problem 3

Again, design a function named 'sum-of-squares' which consumes (listof Number) and produces Number. The function should produce the sum of the squares of all the numbers in the list. But this time, you should use a for-each loop and a single accumulator defined with local. You need not provide a template or a stub. You don't need to provide a signature, purpose, tests, or comments about your accumulator because they would be the same as in your solution to Problem 2.
Problem 4

Assume the following partial data definitions:

;; Natural is one of:
;;   - 0
;;   - (+ 1 Natural)

(define N0 0)
(define N1 42)

;; ListOfString is one of:
;;   - empty
;;   - (cons String ListOfString)

(define LOS1 empty)
(define LOS2 (list "I" "love" "CPSC110"))

Now design a function called 'nth-element' that consumes Natural (n) and ListOfString (los) and produces the nth element of los (using a 0 based index). If los has too few elements, the function produces false.

Treat this as the design of a function operating on 2 complex data. Your design should include a signature, purpose, cross-product table, and tests. You should then provide two working versions of your 'nth-element' function. The first version should be derived directly from your cross-product table, showing all the cases with no simplification. The second version should be your final simplified function definition. You need not provide a function template or a stub. You do not need to explain the simplification steps in your solution.
Problem 5

Start with the two following function designs and design an abstract function from them. Also redefine the two functions as 'one-liners' using the abstract function (for the redefinitions you just need to write the function definition, you do not need to re-write the signature, purpose or tests).

;;; (listof Image) -> Image or false
;;; produce first tall image in loi; false if none is found
(check-expect (fn1 empty) false)
(check-expect (fn1 (list (rectangle 20 10 "solid" "blue")
 (rectangle 10 20 "solid" "red")
 (square 20 "solid" "green")))
 (rectangle 10 20 "solid" "red"))

(define (fn1 loi)
 (cond [(empty? loi) false]
 [else
 (if (tall? (first loi))
 (first loi)
 (fn1 (rest loi)))]))

(define (tall? img)
 (> (image-height img)
 (image-width img)))

;;; (listof Number) -> Number
;;; produce first negative number in lon; 0 if none is found
(check-expect (fn2 empty) 0)
(check-expect (fn2 (list 1 -2 3)) -2)

(define (fn2 lon)
 (cond [(empty? lon) 0]
 [else
 (if (negative? (first lon))
 (first lon)
 (fn2 (rest lon)))]))
Problem 6

Along with this exam, you have received a copy of the incomplete design of a world program named 'balldrop2'. When the balldrop2 program is run, a ball should appear at the upper left-hand corner of the screen, fall to the bottom right-hand corner of the screen, and then disappear off the screen. However, the balldrop2 program has not been completed. Your job is to provide the complete design of the missing functions. You do not have to retain the stub or template, but you must provide everything else. You may not use mutation in the design of your functions.

Write your solutions below. DO NOT WRITE YOUR SOLUTIONS ON THE COPY OF THE PROGRAM THAT HAS BEEN PROVIDED TO YOU AS A SEPARATE DOCUMENT.
Problem 7 - Note that this problem has parts a, b, and c

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 one of
;; - empty
;; - (cons PageName (listof PageName))

;; (listof String) is one of
;; - empty
;; - (cons String (listof String))

(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")))))

Problem 7a

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.
Problem 7b

Write the template for a function operating on PageName. Include any additional template functions for types involved in mutual recursion cycles.
Problem 7c

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. You need not provide a template or a stub. Here are some 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.