Instructions:

1. This is an open book, open notes exam. You may use your CS account to access the book and course notes, including your labs and assignments. You may use DrRacket, but are on your honor not to use the Stepper, Interactions Pane, or Run any programs.

2. Budget your time. There are 75 points on this exam, so you should spend about 1 minute per point (e.g., 15 minutes on a 15 point question). Don’t spend too much time on any one question.

3. Points for each question are indicated below.

4. Please don’t hesitate to ask any questions.

5. Good luck!!!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Russian Dolls Function</td>
<td>15</td>
</tr>
<tr>
<td>2. List of Structs</td>
<td>15</td>
</tr>
<tr>
<td>3. Multiple Lists Function</td>
<td>15</td>
</tr>
<tr>
<td>4. Binary Tree Function</td>
<td>15</td>
</tr>
<tr>
<td>5. Files and Directories</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>75</strong></td>
</tr>
</tbody>
</table>
Problem 1. (15 points)

According to Wikipedia, a Russian doll is “a set of dolls of decreasing sizes placed one inside the other.” Section 9.4 of *How to Design Programs, Second Edition* gives the following structure and Data Definition for a Russian doll:

```
(define-struct layer (color doll))
```

; An RD (short for Russian Doll) is one of:
; - String
; - (make-layer String RD)

Design the function `contains-all?`, which consumes an RD and a color (String) and determines whether the RD contains only dolls of the given color. Follow the Design Recipe, though no need to include a separate template, since one is provided in Section 9.4. Your solution should include a signature, purpose statement, header, and at least four examples (in the form of `check-expects`), and function body.
Problem 2. (15 points)

Recall from lecture 5 our data definition for an animal, and suppose we have a list of animals:

; An animal is either
; - snake
; - dillo
; - ant

; An list-of-animal (LOA) is either
; - '() 
; - (cons animal LOA)

Design the function count-snakes, which consumes a list of animals, and counts the number of snakes. Follow the Design Recipe. Your solution should include a template for a function that consumes a list of animals, fun-for-loa; and a signature, purpose statement, function header, at least four examples (in the form of check-expects), and the function body for count-snakes. You do not need to include a template for an animal, but you will need to use the predicate function for the appropriate animal in the body of your count-snakes.
Problem 3. (15 points)

Given the now familiar Data Definition for a list of numbers:

; A list-of-number (LON) is either
; - ’()
; - (cons number LON)

and the template we saw in Lecture 9 for a function that consumes two lists of the same size:

(define (fun-for-lons lon1 lon2)
  (cond
    [(empty? lon1) ...]
    [(cons? lon1) ...(first lon1)...(first lon2)...
     ...(fun-for-lons (rest lon1) (rest lon2))...]))

Design a function named all-diff? that consumes two LONs of the same size, and determines whether the first number from lon1 is different from the first number from lon2, the second number from lon1 is different from the second number from lon2, and so on. Follow the Design Recipe, though no need to include a separate template, since one is provided above. Your solution should include a signature, purpose statement, header, examples (in the form of at least three check-expects), and function body. Hint: two numbers are different if they are not the same.
Problem 4. (15 points)

Here is a data definition for a binary tree (which is like our original rumor mill, except this tree contains node structs instead of gossiping person structs):

(define-struct node (value left right))

; A binary-tree (BT) is either
; - '()
; - (make-node Number BT BT)

Figure 1 below depicts a binary tree. In Figure 1, the circles represent node structs, the number in each circle represents the node's value, and the lines below each circle lead to the left and right BTs for each node. (Nodes 1, 3, 4, and 6 have '() left and right fields, which are not shown in the figure.)

Figure 1: A binary tree

(a) (7 points) Using the data definition given for a BT, translate Figure 1 into its Racket representation. Hint: it’s easiest to build the BT one level at a time, starting from the bottom. That is, give definitions for nodes 1, 3, 4, and 6 first, then use those names for defining nodes 2 and 5, etc.
b) (8 points) Develop the predicate function `contains-odd?`, which consumes a binary tree (BT), and determines whether the given BT contains an odd number. **For full credit, be sure to include the following hand-in artifacts from the Design Recipe:** template `fun-for-bt`, signature/purpose/header, examples (you should have at least three, based on the data definition for BT, and the nature of this function), and last but not least, the function body.
Problem 5. (15 points) Recall (from lecture 12) our intertwined data definitions for files and directories:

; A directory is
; - (make-dir string LOFD)
(define-struct dir (name content))

; A file is
; - (make-file string number)
(define-struct file (name size))

; A file-or-directory is either
; - file
; - directory

; A LOFD is either
; - '()
; - (cons file-or-directory LOFD)

Design four versions of the function count, one for each of the four intertwined data definitions. Collectively these four functions count the total number of files and directories in the directory tree.

To help you get started, I’ve provided the signatures for each of the four functions. You may find it helpful to use the corresponding templates provided in the Lecture 12 notes. You need only provide purpose statements and the functions for this question. If you’re not sure how to write the functions, provide the templates for each function for partial credit.

; count-for-dir : directory -> number
;

; count-for-file : file -> number
;

; count-for-fod : file-or-directory -> number
;

; count-for-lofd? : LOFD -> number
;