Name: __________________________

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!!!
Problem 1. (15 points)

Here are two examples of a list of numbers (LONs), expressed using list abbreviations:

```
(define odds (list 1 3 5 7 9))
(define evens (list 2 4 6 8 10))
```

a) (4 points) Rewrite the definitions above for **odds** and **evens** using **cons** expressions—no list abbreviations.

b) (5 points) Evaluate the following expressions. For expressions that evaluate to a list, you may use list abbreviations or **cons** expressions, whichever you prefer.

```
(first evens)
(rest odds)
(first (rest (rest evens)))
(empty? (rest (rest (rest (rest (rest evens))))))
(cons? (rest (rest (rest (rest (rest odds))))))
```

c) (6 points) Write expressions using **first**, **rest**, **cons**, `'()`, and/or list abbreviations (but no numbers), and the two lists defined above, to produce the following:

- list containing only the last number from the **odds** list
- the fourth element from the **evens** list (not a list, just the element)
- list containing the second element of both lists (i.e., 3, 4)
Problem 2. (15 points)

Write a function named `contains-odd?` that consumes a list of numbers, and determines whether that list contains an odd number. (You should use the existing function `odd?` in your solution.) **Follow the Design Recipe step by step.** You will receive points for each hand-in artifact from the design recipe you include in your solution. For full credit, be sure to include a data definition (3 points), template (3 points), signature/purpose statement/header (3 points), examples (3 points), and last but not least, the function body (3 points).
Problem 3. (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 persons to gossip to):

```racket
(define-struct node (value left right))
```

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

Figure 1 below depicts a binary tree. In Figure 1, the circles represent node structs, and the lines below each circle represent the left and right BTs for each node. (Nodes 4, 5, 6, and 7 have "null" left and right fields, which are not shown in the figure.)

![Figure 1: A binary tree](image)

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, define names for nodes 4, 5, 6, and 7 first, then use those names for defining 2 and 3, etc.*
Develop the function \texttt{sum-odds}. It consumes a binary tree (BT) and produces the sum of all the odd numbers in the tree. You should use the existing function \texttt{odd?} in your solution. \textbf{For full credit, be sure to include the following remaining hand-in artifacts from the Design Recipe:} template \texttt{fun-for-bt}, signature/purpose/header, examples (you should have at least two, based on the data definition for BT), and last but not least, the function body.
Problem 4. (15 points)

According to Wikipedia, a Russian doll is “a set of dolls of decreasing sizes placed one inside the other.” Section 10.4 of How to Design Programs, Second Edition gives the following struct and data definition for a Russian doll:

```
(define-struct layer (color doll))
; An RD (russian doll) is one of:
; - String
; - (make-layer String RD)
```

To see a picture of Russian dolls, or read more about designing functions for this type of data, including the template, scroll down to section 10.4 of Part II of HtDP/2e at URL: http://www.ccs.neu.edu/home/matthias/HtDP2e/part_two.html

(7 points) Design the function `inner`, which consumes an RD and produces the (color of the) innermost doll. Follow the Design Recipe, though no need to include a separate template, since its provided in section 10.4.

(8 points) Design the function `contains-color?`, which consumes an RD and a color (e.g., "red") and determines whether the given RD contains a doll of the given color. Follow the Design Recipe.
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)

Develop four related functions, larger-than-dir, larger-than-file, larger-than-file-or-directory, and larger-than-lofd, that correspond to these intertwined data definitions. Each of these functions should determine whether the given directory/file/file-or-directory/LOFD contains a file larger than the given size (a number).

To help you get started, I’ve provided the signatures for each function. 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.

; larger-than-dir : directory num -> bool

; larger-than-file : file num -> bool

; larger-than-file-or-directory : file-or-directory num -> bool

; larger-than-lofd : LOFD num -> bool