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!!!

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>received</td>
</tr>
<tr>
<td>1. List Expressions</td>
<td></td>
</tr>
<tr>
<td>2. List Function</td>
<td></td>
</tr>
<tr>
<td>3. Binary Tree</td>
<td></td>
</tr>
<tr>
<td>4. The Cat in the Hat Comes Back</td>
<td></td>
</tr>
<tr>
<td>5. Files and Directories</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (15 points)

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

```
(define numbers (list 1 2 3 4 5))
(define letters (list "a" "b" "c" "d" "e"))
```

a) (4 points) Rewrite the definitions above for `numbers` and `letters` 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 (rest numbers))
(rest (rest letters))
(first (rest (rest (rest (rest letters))))))
(empty? (rest (rest (rest (rest numbers))))))
(cons? letters)
```

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

- list containing "a" from the `letters` list followed by 1 from the `numbers` list

- the third element from the `letters` list (not a list, just the element)

- list containing 2 from the `numbers` list followed by "b" from the `letters` list
Problem 2. (15 points)

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

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

Design a function named sum-odds that consumes a LON, and produces the sum of the odd numbers in the list. (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 signature/purpose statement/header (3 points), template (4 points), examples (4 points), and last but not least, the function body (4 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 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, 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

\[\text{\includegraphics{binary-tree.png}}\]

a) (7 points) Using the data definition given for a BT, translate Figure 1 into its Racket representation. \textit{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 function sum-evens. It consumes a binary tree (BT) and produces the sum of all the even numbers in the tree. You should use the existing function even? in your solution. 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 two, based on the data definition for BT), and last but not least, the function body.
Problem 4. (15 points)

*The Cat in the Hat Comes Back* is a Dr. Seuss book that your professor loved to read to his children when they were younger. This sequel to the original *Cat in the Hat* reveals a new recursive quality to the Cat in the Hat! When the Cat takes off his hat, Little Cat A is underneath, and under Little Cat A’s hat is Little Cat B, and so on until we find Little Cat Z. Under Little Cat Z’s hat, however, is something else: Voom! If you don’t know what Voom is, I recommend you read the book! (Some of the greatest books for adults are disguised as children’s books.) We can represent The Cat with the following Data Definition:

```
(define-struct cat (letter hat))
; A Cat is either:
; - "Voom"
; - (make-cat String Cat)
```

a) (3 points) Define three (3) examples of Cats. These are data examples only. Not functions. The first Cat should just be "Voom". The second Cat should have "Voom" in its hat. The third Cat should have the second Cat in its hat. (i.e., the second and third Cats will both be cat structs).

b) (5 points) Design the template for a function that consumes a Cat. Your template function should be named fun-for-cat. Be sure to include an appropriate signature and purpose statement for your template function. You will use this template to design a function next.
c) (7 points) Design the function `hat-depth`, which consumes a Cat and determines how many hats "Voom" is under. Use the template you just designed, `fun-for-cat`, to help you get started. Follow the Design Recipe. Use your three examples of Cats to create `check-expects` to test your `hat-depth` function.
Problem 5. (15 points) Recall (from lecture 11) our intertwined data definitions for a more realistic rumor mill:

```scheme
;; A list-of-gossip is either
;; - '()
;; - (cons gossip list-of-gossip)
;; A gossip is a
;; (make-gossip image list-of-gossip)
(define-struct gossip (who nexts))
```

Design the function `rumor-delay` which takes a gossip and determines the maximum number of days required for a rumor to reach everyone, assuming that each person waits a day before passing on a rumor.

More specifically, you will need to develop two related functions, `rumor-delay-gossip` and `rumor-delay-list-of-gossip`, that correspond to the above two intertwined data definitions. Both of these functions should determine the number of days required for a rumor to reach everyone.

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 11 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.

*Hint: you may find it convenient to use the built-in function, max, which finds the max of two given values.*

```scheme
; rumor-delay-gossip : gossip -> number
;
; rumor-delay-list-of-gossip : list-of-gossip -> number
;```