CS 2500 Exam 2—Fall 2016
Your Beloved CS 2500 Instructors

You are attending the lecture of ____________________________

• The exam is a one-hour exam. To accommodate everyone’s needs for
time and space, the instructors will stay for three hours.

• Write down the answers in the space provided.

• You may use the usual primitives and expression forms, including
those suggested in hints; for everything else, define it.

• The phrase “design a function” means that you should apply the de-
sign recipe.

You do not have to spell out examples as test cases (with check-
expect and friends), but you are welcome to do so.

When a problem asks for a complete function, you are not required
to provide a template. But, if you elect to skip the template step, be
prepared to struggle with the development of the function.

• Some basic test taking advice: Before you start answering any prob-
lems, read every problem, so your brain can be thinking about the
harder problems in background while you knock off the easy ones.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max. Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/ 10</td>
</tr>
<tr>
<td>2</td>
<td>/ 16</td>
</tr>
<tr>
<td>3</td>
<td>/ 6</td>
</tr>
<tr>
<td>4</td>
<td>/ 14</td>
</tr>
<tr>
<td>Total</td>
<td>/ 46</td>
</tr>
</tbody>
</table>
Problem 1 Design the function remove-both. It consumes a string \( n \) and a PB, the data representation of phone books designed by your current partner. The result is a PB but with all occurrences of \( n \) and the following phone number removed.

; A PB is one of:
; -- '()
; -- (cons String (cons Number PB))
; interpretation A phone book such as
; (cons "Alan" (cons 617738.1212 pb))
; means "Alan"'s phone number is 617738.1212,
; and the rest of the phone book is pb
**Problem 2** Design the function `drop`. It consumes and produces a list of Posns. Each Posn whose y coordinate is larger than 200 is removed. All other y coordinates are increased by 3.

(a) Develop the signature for `drop`, its purpose statement, and examples. You may assume the standard definition of *Posn*.
(b) Use the existing abstractions (2e: figures 91 and 92; 1e: figure 57, p. 313; reproduced as figures 1 and 2 at the end of the exam) to complete the design of drop.
(c) Complete the design of the function *without* the use of existing abstractions ("loops").
Problem 3 Design count-wings. The function consumes a Butterfly and counts the pairs of wings that surround its body. You may assume the standard definition of \( N \) (natural numbers).

\[
\begin{align*}
\text{(define LEFT-WING "\(\)\")} \\
\text{(define RIGHT-WING \\ "\(\)\")}
\end{align*}
\]

; A Butterfly is one of:
; -- "body"
; -- (list LEFT-WING Butterfly RIGHT-WING)

Not every left-wing, right-wing animal is about politics.
Problem 4 Design the function cleanse, which consumes an Enumeration and removes all bad strings. To simplify the problem, we assume that there is only one bad string: "@#$%".

```scheme
(define-struct bullets (loi))
(define-struct points (loi))
(define-struct item (low))

;; An Enumeration is one of:
;; -- (make-bullets LoI) ;; bulletized items
;; -- (make-points LoI) ;; numbered points
;;
;; An LoI is a [List-of Item]
;;
;; An Item is a structure: (make-item LoW)
;;
;; An LoW is a [List-of Word]
;;
;; A Word is one of:
;; -- String
;; -- Enumeration
;;
;; interpretation An Enumeration is a generic data representation of HTML, LateX, etc nested, itemized lists.
```
; [X] N [N -> X] -> [List-of X]
; constructs a list by applying $f$ to 0, 1, ..., (sub1 n)
; (build-list n f) == (list (f 0) ... (f (- n 1)))
(define (build-list n f) ...)

; [X] [X -> Boolean] [List-of X] -> [List-of X]
; produces a list from those items on $lx$ for which $p$ holds
(define (filter p lx) ...)

; [X] [List-of X] [X X -> Boolean] -> [List-of X]
; produces a version of $lx$ that is sorted according to $cmp$
(define (sort lx cmp) ...)

; [X Y] [X -> Y] [List-of X] -> [List-of Y]
; constructs a list by applying $f$ to each item on $lx$
; (map f (list x-1 ... x-n)) == (list (f x-1) ... (f x-n))
(define (map f lx) ...)

; [X] [X -> Boolean] [List-of X] -> Boolean
; determines whether $p$ holds for every item on $lx$
; (andmap p (list x-1 ... x-n)) == (and (p x-1) ... (p x-n))
(define (andmap p lx) ...)

; [X] [X -> Boolean] [List-of X] -> Boolean
; determines whether $p$ holds for at least one item on $lx$
; (ormap p (list x-1 ... x-n)) == (or (p x-1) ... (p x-n))
(define (ormap p lx) ...)

Figure 1: ISL’s abstract functions for list-processing (1)
(foldr + 0 '(1 2 3 4 5))
== (+ 1 (+ 2 (+ 3 (+ 4 (+ 5 0))))))
== (+ 1 (+ 2 (+ 3 (+ 4 5))))
== (+ 1 (+ 2 (+ 3 9)))
== (+ 1 (+ 2 12))
== (+ 1 14)

(foldl + 0 '(1 2 3 4 5))
== (+ 5 (+ 4 (+ 3 (+ 2 (+ 1 0)))))
== (+ 5 (+ 4 (+ 3 (+ 2 1))))
== (+ 5 (+ 4 (+ 3 3)))
== (+ 5 (+ 4 6))
== (+ 5 10)

Figure 2: ISL's abstract functions for list-processing (2)