Student Name: _____

CS 2500/Accelerated Exam 1—Fall 2018

Amal Ahmed

October 17, 2018

- We will not answer questions during the exam. If you believe a problem statement is ambiguous, choose *any* non-trivial interpretation.
- Write down the answers in the space provided, including the back of the given spaces and pages marked "intentionally left blank".
- You may use the paper copy of the book, your notes, and design-recipe cards.
- You may *not* use any electronic gadgets (for example, watches, google glasses, phones, tablets, laptops). Any use of an electronic gadget will lead to immediate expulsion from the exam and class.
- You may use all the definitions, expressions, and functions found in ISL+. Define everything else.
- The phrase "design a program" means that you should apply the data and function design recipes. You may write "`c-e`" as shorthand for `check-expect`.
- Unless a problem requests a solution that does not use ISL abstractions, you may use these abstractions. Similarly, unless a problem demands a solution that uses the abstractions of ISL, you do not have to use these abstractions.
- Basic test-taking advice: Before you start answering any problems, read *every* problem, so your brain can be thinking about the harder problems in the background while you knock off the easy ones.

Problem	Max. Points
1	10
2	18
3	11
4	8
5	13
Total	/ 60
Extra Credit	4

Exercise 1 (10 points)

Ahead of the November elections, you've been hired as a consultant for a political campaign. The campaign keeps track of contributions using the following data definition:

```
; An LOC (list of campaign contributions) is
; one of:
; – empty
; – (cons Contrib LOC)

(define-struct contrib [donor amount])
; A Contrib (campaign contribution) is one of:
; – Number
; – (make-contrib String Number)
; interpretation A simple number represents an
; anonymous contribution, while a
; (make-contrib d amt) represents a
; contribution of amt dollars from donor d.
```

The rules of campaign finance declare that an individual may only donate $2700 to a given campaign. Furthermore, anonymous contributions are limited to $50.

Your first task is to design a program, `any-bad-contrib?`, that checks, given a list of contributions, if any of the contributions on the list are illegal ones. Follow the structural design recipe. Do not use ISL abstractions.

intentionally left blank

Exercise 2 (18 points)

Donors can try to get around the rules of campaign finance (see the previous problem) by making multiple donations. The campaign you are working for would like to ensure that none of their donors have violated the law by contributing a total over $2700. Your next task as consultant is to design a program, `any-bad-donors?`, that takes a list of campaign contributions and checks if the total contributions of any of the named donors on the list exceeds $2700. You may reuse data examples you defined in Problem 1 for tests.

intentionally left blank

Exercise 3 (11 points)

Design the function `remove-duplicates`. It should take

1. a list `l`
2. a function `eq?` that consumes two inputs (of the same type as elements of `l`) and checks if they are equal

and remove all duplicate elements from the input list.

It should satisfy the following test. Note the order of the elements in the output list.

```
(check-expect
  (remove-duplicates (list 1 2 3 2 4 3 2) =)
  (list 1 2 3 4))
```

intentionally left blank

Exercise 4 (8 points)

Design the function `map` that, just like the `map` function in ISL, consumes a function `f` and a list `l`, and produces a list with the results of applying `f` to each element of the input list `l`. You must define the function using just `foldr`, as follows:

```
(define (map f l)
  (foldr ...))
```

Exercise 5 (13 points)

Design the common abstraction, including signature, for the following two function definitions, `long-films` and `head-1st-quadrant`. Then re-implement both functions using your newly defined abstraction. You do not need to re-write their signatures, purpose statements, or tests.

long-films

```
(define-struct film [title runtime])
; A Film is a (make-film String Number)
; It represents a film's name and runtime in minutes

; long-films : [List-of Film] -> [List-of String]
; The titles of films longer than 2 hours
(define (long-films lof)
  (cond [(empty? lof) empty]
        [(cons? lof)
         (if (long-film? (first lof))
             (cons (film-title (first lof)) (long-films (rest lof)))
             (long-films (rest lof)))]))
(check-expect (long-films empty) empty)
(check-expect (long-films (list (make-film "LotR" 178)
                                (make-film "Casablanca" 102)))
              (list "LotR"))

; long-film? : Film -> Boolean
; Is this film longer than 2 hours?
(define (long-film? film)
  (> (film-runtime film) 120))
(check-expect (long-film? (make-film "LotR" 178)) #true)
(check-expect (long-film? (make-film "Casablanca" 102)) #false)
```

head-1st-quadrant

```
; Snake = NESegs
; NESegs (non-empty segments) is the list of segments of the
; snake's body; first element in the list is the head

; An NESegs (non-empty segments) is one of:
; (cons Posn empty)
; (cons Posn NESegs)

; head-1st-quadrant : [List-of Snake] -> [List-of Snake]
; Return only the snakes whose heads lie in the first quadrant
(define (head-1st-quadrant losnk)
  (cond [(empty? losnk) empty]
        [(cons? losnk)
         (if (first-quadrant? (first (first losnk)))
             (cons (first losnk) (head-1st-quadrant (rest losnk)))
             (head-1st-quadrant (rest losnk)))]))
(check-expect (head-1st-quadrant empty) empty)
(check-expect (head-1st-quadrant
               (list (list (make-posn 1 1) (make-posn 0 1))
                     (list (make-posn 0 1) (make-posn 1 1))))
              (list (list (make-posn 1 1) (make-posn 0 1))))


; first-quadrant? : Posn -> Boolean
; Is this posn in the first quadrant?
(define (first-quadrant? p)
  (and (> (posn-x p) 0) (> (posn-y p) 0)))
(check-expect (first-quadrant? (make-posn 1 1)) #true)
(check-expect (first-quadrant? (make-posn 1 0)) #false)
(check-expect (first-quadrant? (make-posn 0 1)) #false)
```

intentionally left blank