Student Name: ______________________________

CS 2500/Accelerated Exam 2—Fall 2017

Matthias Felleisen

November 15, 2017

- The exam is a **one-hour** exam.
- We will not answer any questions during the exam. If you believe a problem statement is ambiguous, write down your thoughts and choose *any* non-trivial interpretation.
- Write down the answers in the space provided, including the back of the given spaces. If you need more space, ask for another blank exam.
- You may use the paper copy of the book or your notes.
- 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 ISL+. Define everything else.
- Unless a problem requests a solution that does not use the abstractions of ISL+—see figures 1 and 2 on the back of this page—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.

Problem	Max. Points
1	10
2	12
3	21
4	9
Total	/ 52

```
; [X] N [N -> X] -> [Listof 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] [Listof X] -> [Listof X]
; produces a list from those items on lx for which p
holds
(define (filter p lx) ...)

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

; [X Y] [X -> Y] [Listof X] -> [Listof 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] [Listof 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] [Listof 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)

```
; [X Y] [X Y -> Y] Y [Listof X] -> Y
; (foldr f b (cons x-1 ... (cons x-n '()) ..)) ==
; (f x-1 ... (f x-n b))
(define (foldr f b lx) ...)

(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)

; [X Y] [X Y -> Y] Y [Listof X] -> Y
; (foldr f b (cons x-1 ... (cons x-n '()) ..)) ==
; (f x-n ... (f x-1 b))
(define (foldl f b lx) ...)

(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)

Problem 1 Design `echo` using one of the existing abstractions. The function consumes a list of `String`s. Its result contains every String followed by a `String` that represents the length of the original one. **Hint** `number->string` comes in handy.

10pts.

Show all steps of the template design recipe for using "loops."

Problem 2 Design `good?`. The function consumes a Signature and makes sure that every symbol that occurs in the Signature belongs to BaseType. *12pts.*

```
; A BaseType is one of:
; -- 'Number
; -- 'String
; -- 'Symbol
; -- 'Boolean
; -- 'Image

(define-struct -> [domain range])
(define-struct union [parts])
; A Signature is one of:
; -- Symbol
; -- (make--> List-of-Signatures Signature)
; -- (make-union List-of-Signatures)
;
; A List-of-Signatures is one of:
; -- (cons Signature '())
; -- (cons Signature List-of-Signatures)
```

intentionally left blank

Problem 3 Design the function `depth` for binary trees: *21pts.*

```
(define-struct leaf [info])
(define-struct node [left info right])
; A [BT X] (binary tree over X) is one of:
; -- (make-leaf Symbol)
; -- (make-node [BT X] X [BT X])
; interpretation binary trees with nodes that
; carry some X-kind of value and Symbols as leaves
```

It consumes a [BT String] and replaces each `String` in a node with the distance to the “root” of the given tree.

(a) Design `depth` **without** using an accumulator.

(b) Explain in one complete English sentence why using an accumulator is a good idea.

(c) Design `depth` **using** an accumulator-based approach.

Reuse the signature and examples from (a).

intentionally left blank

Problem 4 Design the function `layer-cake`. It consumes an LC and counts how many `'layer`s are wrapped around `'cake`. *8pts.*

```
; An LC is one of:
; -- 'cake
; -- (cons 'layer (cons LC '()))
; interpretation an LC represents a layer cake
```