CSSE 304 Exam #1 Winter 2021-2022 Section 02 (11:00) 03 (2:00)

Name

Key

You must turn in Part 1 before you use your computer for anything. During the entire exam you may not use email, IM, phone, tablet, headphones, ear buds, or any other communication device or software. Except where specified, efficiency and elegance will not affect your scores, provided that I can understand your code.

On both parts, assume that all input arguments will be of the correct types for any procedure you are asked to write; you do not need to check for illegal input data.
Mutation is not allowed in code that you write for this exam, except where noted.

Part 1, written. Allowed resources: Writing implement.
Suggestion: Spend no more than 40 minutes on this part, so that you have a lot of time for the computer part. 30 minutes is ideal.

Built-in procedures & syntax that are sufficient for this paper part of this exam:

Procedures:
Arithmetic: +, - , *, /, modulo, max, min, =, <, ≤, >, ≥
Predicates and logic: not, eq?, equal?, null?, zero?, procedure? positive?, negative?, pair?, list?, even?, odd?, number?, symbol?, integer?, member
Lists: cons, list, append, length, reverse, set-car!, set-cdr!, car, cdr, cadr, cddr, etc.
Functional: map, apply, andmap, ormap, filter
Homework: Any procedure that was assigned for A01-A08.

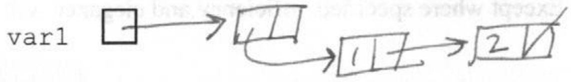
Syntax:
lambda, including (lambda x …) and (lambda (x y . z) …),
define, if, cond, and, or, let, let*, letrec, named let, begin, set! (You may not use mutation in your code unless a specific problem says you can).

Do not start this exam before instructed to do so. Do write your name on both pages as soon as you get the exam.

1. **(6 points)** Consider the execution of the code below. Draw the box-and-pointer diagrams that represent the results of the `defines`. Then show what Scheme would output from the execution of each of the last three expressions.
Be careful! "Almost correct" answers will usually receive no partial credit.

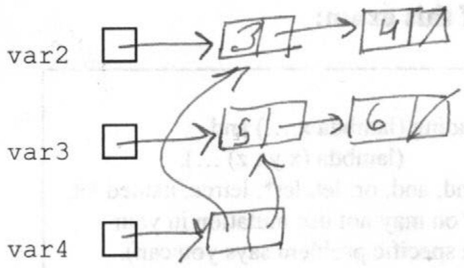
1a (2 points).

```
(define var1 '((1 2)))
```



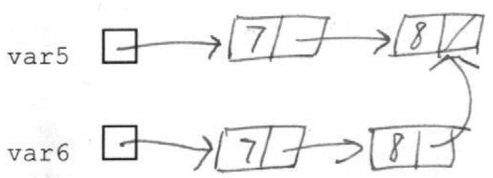
1b (2 points).

```
(define var2 '(3 4))
(define var3 '(5 6))
(define var4 (cons var2 var3))
```



1c (2 points).

```
(define var5 '(7 8))
(define var6 (append var5 (cdr var5)))
```



2. **(5 points)** Though we often curry bigger functions into 1-parameter functions, there's nothing magical about 1-parameter functions. Write a function double-curry4 that takes a 4 parameter function and curries it into a curried function that takes two parameters. See the usage below:

```
(define dc-list (double-curry4 list))
(define partial (dc-list 1 2))
(partial 3 4) ;; returns (1 2 3 4)
```

(define double-curry4

```
  (lambda (proc)
     (lambda (a b)
        (lambda (c d)
          (proc a b c d))))
```

3. **(5 points)** Imagine I have a list of some combination of symbols and numbers. I want to compute the sum of that list, ignoring any symbols. For example:

`(sum-mixed '(1 a 3 10 q))` yields 14

Write sum-mixed using some combination of map, filter, and apply. Do not use any recursion or looping constructs in your solution.

(define sum-mixed

```
  (lambda (lst)
     (apply + (filter number? lst))))
```

Name____________________

4. **(4 points)** Write a definition for a procedure make-counter that returns a procedure. When this procedure is first called, it should return 1. Then the second call returns 2, etc. See the code example to see how make-counter is used. It should be possible to create any number of counters, each with their own count. Mutation is allowed on this question.

```
(define c1 (make-counter))
(define c2 (make-counter))
(c1) ; returns 1
(c1) ; returns 2
(c2) ; returns 1
```

```
(define make-counter
  (lambda ()
     (let ((count 0))
        (lambda ()
           (set! count (add1 count))
           count)))
```

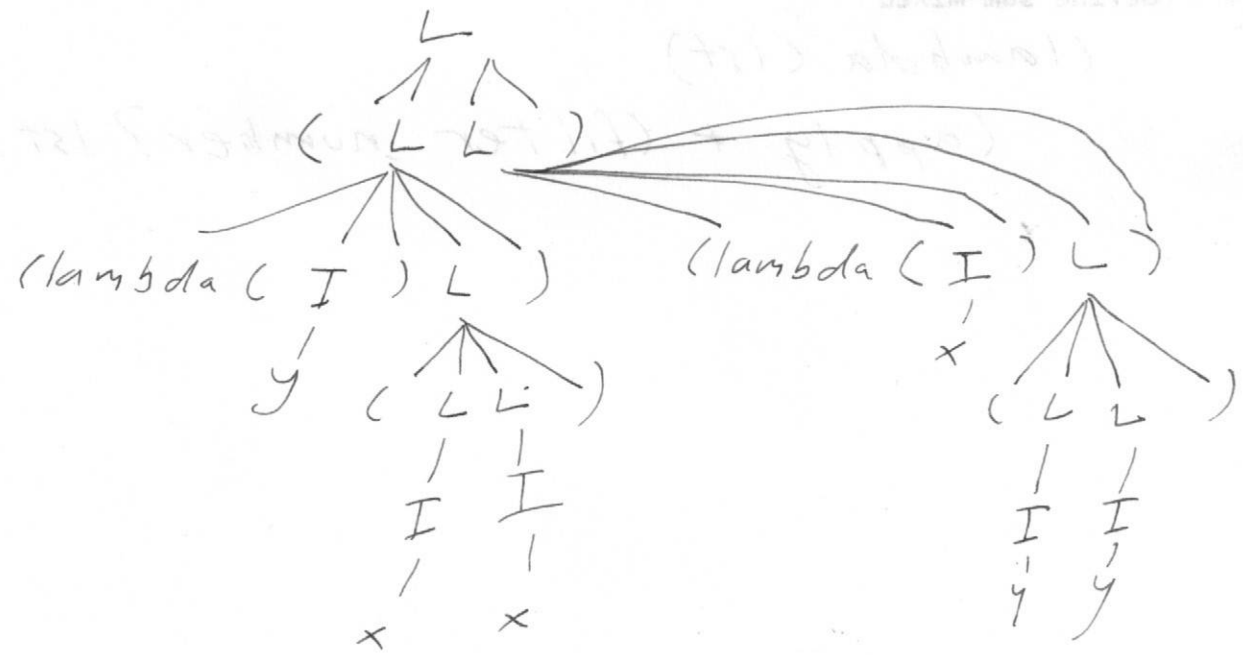
```
<LcExpr> ::=
 <identifier> |
 (lambda (<identifier>) <LcExpr>) |
 ( <LcExpr> <LcExpr> )
```

5. (5 points) Our original grammar for lambda-calculus expressions:

Consider the expression `((lambda (y) (x x)) (lambda (x) (y y)))`

(a) **(2 points)** In that expression, which variables occur bound? ____________ occur free? x y

(b) **(3 points)** Draw the derivation tree for that expression. To make it easier to draw this (as I did on the whiteboard in the Day 11 class), you are allowed to write **L** in place of `<LcExpr>`, **λ** in place of `lambda`, and **id** in place of `<identifier>`.


L
(L L)
(lambda (I) L)
y
(L L)
I I
x x
(lambda (I) L)
x
(L L)
I I
y y