Your name ______________________________ A random five-digit number: ________

Circle the last two letters of your login (`cs61a-xx`)

`a b c d e f g h i j k l m n o p q r s t u v w x y z 1 2 3`

`a b c d e f g h i j k l m n o p q r s t u v w x y z`

This exam is worth 70 points, or about 23% of your total course grade. The exam contains 14 questions.

This booklet contains 14 numbered pages including the cover page. Put all answers on these pages, please; don't hand in stray pieces of paper. This is an open book exam.

When writing procedures, don't put in error checks. Assume that you will be given arguments of the correct type.

If you want to use procedures defined in the book or reader as part of your solution to a programming problem, you must cite the page number on which it is defined so we know what you think it does.

***** IMPORTANT *****

Check here if you are one of the people with whom we arranged to replace a missed/missing exam with other exam scores: _____

***** IMPORTANT *****

If you have made grading complaints **that have not yet been resolved**, put the assignment name(s) here:

READ AND SIGN THIS:

I certify that my answers to this exam are all my own work, and that I have not discussed the exam questions or answers with anyone prior to taking this exam.

If I am taking this exam early, I certify that I shall not discuss the exam questions or answers with anyone until after the scheduled exam time.

____

1–2	/7
3	/4
4	/7
5-6	/7
7	/6
8	/4
9	/6
10	/4
11	/6
12	/6
13	/7
14	/6
total	/70

Question 1 (3 points):

What will the Scheme interpreter print in response to each of the following expressions? If any expression results in an error or infinite loop, just write "Error".

```
> (define (foo x)
    (define count (+ x count))
    count)
> (define count 0)
> (foo 3)
```

____

```
> (foo 4)
```

____

```
> count
```

____

Question 2 (4 points):

(a) Give an example of something that is a pair but not a list. If there is no such thing, write "impossible".

(b) Give an example of something that is a list but not a pair. If there is no such thing, write "impossible".

(c) What will the Scheme interpreter print in response to the second expression below? If it results in an error or infinite loop, just write "Error".

```
> (define my-stream (cons-stream 1 my-stream))
> (list (pair? my-stream) (list? my-stream))
```

____

Question 3 (4 points):

What is the order of growth in time of each of the following procedures? Assume "N" is the length of the list.

```
(define (pinky lst)
  (if (null? lst)
      0
      (+ (accumulate + 0 lst) (pinky (cdr lst)))))
```

______ $\Theta(1)$ ______ $\Theta(N)$ ______ $\Theta(N^2)$ ______ $\Theta(N^3)$ ______ $\Theta(2^N)$

```
(define (brain lst)
  (if (null? lst)
      0
      (+ (+ (car lst) (brain (cdr lst)))
         (brain (cdr lst)) )))
```

______ $\Theta(1)$ ______ $\Theta(N)$ ______ $\Theta(N^2)$ ______ $\Theta(N^3)$ ______ $\Theta(2^N)$

Question 4 (7 points):

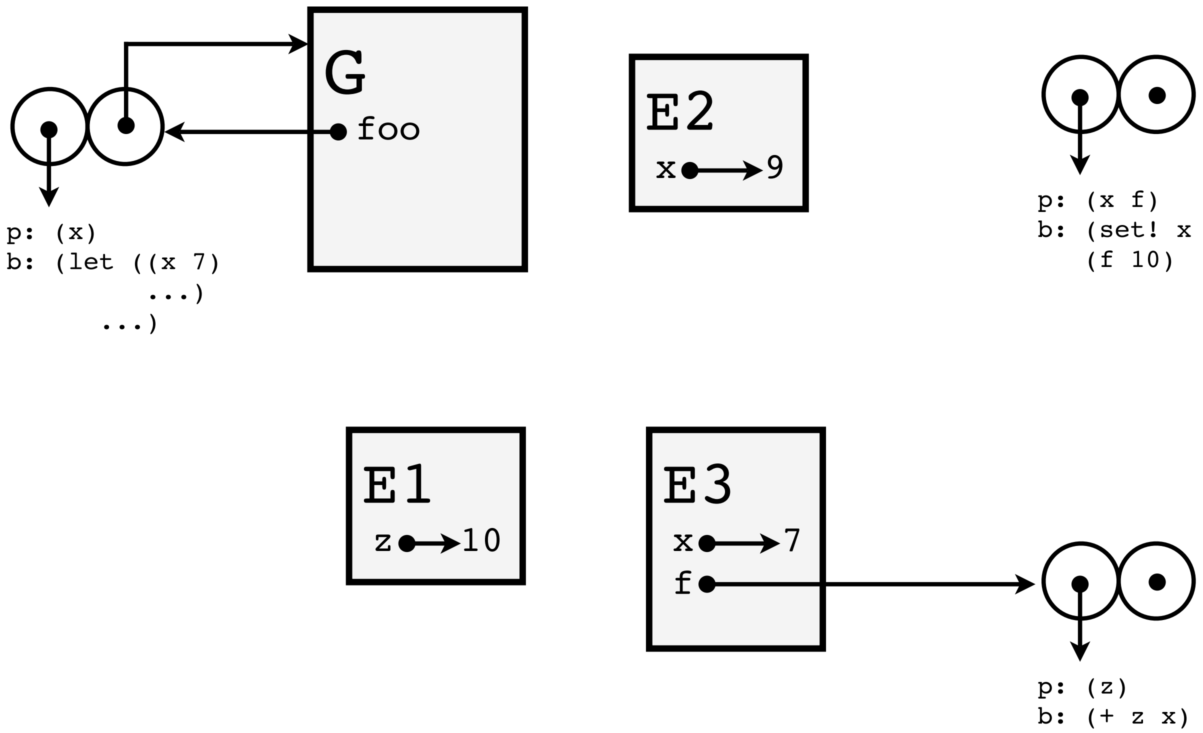
Consider the following code:

```
STk> (define (foo x)
       (let ((x 7) (f (lambda (z) (+ z x))))
         (set! x 3)
         (f 10) ))

STk> (foo 9)
```

____

Running it will create the following environment diagram. Complete the diagram by adding in five missing arrows and the result of the `set!`, then fill in the result of the final expression.



Question 5 (3 points):

Given the following code:

```
(define (foo x y)
  (bar 1 y))

(define (bar a b)
  (* x y))

(foo 3 4)
```

What is the result of the final expression using lexical scope? ______________

What is the result of the final expression using dynamic scope? ______________

(If any expression causes an error, just write "Error".)

Which does Scheme use? ______ Lexical scope ______ Dynamic scope

Question 6 (4 points):

Pick the *best* answer for the questions below.

(a) Ben Bitdiddle is able to send chat messages to Alyssa P. Hacker, but he isn't getting the replies she sends. Which of the following is most likely the problem?

______ His client didn't set a callback.

______ His client didn't finish the three-way handshake.

______ The server isn't sending out updated client-lists when clients log on.

______ The server isn't forwarding chat messages to their destinations.

(b) Which of the following fixes for the Therac-25 would have prevented the most accidents?

______ Add hardware interlocks, like the earlier Therac-20.

______ Log any software-detected abnormalities.

______ Ask operators to confirm high radiation doses.

______ Use serializers to prevent concurrency issues.

Question 7 (6 points):

(a) Consider the following code:

```
> (define cereal-1 (make-serializer))
> (define cereal-2 (make-serializer))
> (define z 10)
> (parallel-execute (cereal-1 (cereal-2 (lambda () (set! z (+ z 20)))))
                    (cereal-2 (cereal-1 (lambda () (set! z 42)))) )
```

Which of the following problems are possible with the code above?

Y ____ N ____ Incorrect results

Y ____ N ____ Deadlock

Y ____ N ____ Inefficiency (missed opportunity for parallelism)

(b) Consider the following code:

```
> (define x 5)
> (define y 6)
> (define x-serializer (make-serializer))
> (define y-serializer (make-serializer))

> (parallel-execute
    (x-serializer (y-serializer (lambda () (set! x (+ x 5)))))
    (x-serializer (y-serializer (lambda () (set! x (* x x)))))
    (x-serializer (y-serializer (lambda () (set! y (+ y 2))))) )
```

Which of the following problems are possible with the code above?

Y ____ N ____ Incorrect results

Y ____ N ____ Deadlock

Y ____ N ____ Inefficiency (missed opportunity for parallelism)

Your five-digit number: ______________

Question 8 (4 points):

Consider the following input to the **lazy evaluator**:

```
(define count 0)

(define (akbar arg)
  (let ((hoho count))
    (set! count 3)
    (set! hoho arg)
    hoho))

(define (jeff)
   (set! count 5)
   42)

(define binky (akbar (jeff)))
```

After evaluating these expressions, what is the value of `count`? ______________

What is the value of `binky`? ______________

(If any expression generates an error or infinite loop, just write "Error". If either value is a promise, just write "Promise".)

Question 9 (6 points):

Consider the following MapReduce query:

```
(define (count-words input)
  (list (make-kv-pair (kv-key input) (length (kv-value input))) ))

(mapreduce count-words + 0 "/shakespeare")
```

The result is a stream of key-value pairs, where the keys are the names of Shakespeare plays, and the values are the number of words in the play.

```
((a-lovers-complaint . 2568) (a-midsummer-nights-dream . 17608) ...)
```

(a) Change **either** the mapper **or** the reducer (but not both) to find the length of the longest line in each play. (You can use an existing Scheme primitive, or write an entirely new procedure. If you change the reducer, you can also change the base case.) Show your new call to `mapreduce`.

(b) Change **either** the mapper **or** the reducer (but not both) to count the number of times the word "thou" appears in each play. (You can use an existing Scheme primitive, or write an entirely new procedure. If you change the reducer, you can also change the base case.) Show your new call to `mapreduce`.

Question 10 (4 points):

The procedure `largest-value` takes a list of key-value pairs and returns the **pair** with the largest **value**:

```
> (define assoc-list (list (make-kv-pair a 1)
                           (make-kv-pair b 42)
                           (make-kv-pair c 9005) ))
> (largest-value assoc-list)
(c . 9005)
```

You may assume the argument to `largest-value` is never the empty list.

Complete this implementation of `largest-value` by defining `lv-helper`.

```
(define (largest-value assoc-list)
  (accumulate lv-helper
              (car assoc-list)
              (cdr assoc-list) ))
```

Question 11 (6 points):

The function `check-children` is supposed to return `#t` if every node in a Tree has a datum equal to its number of children. Fix all errors and data abstraction violations (DAVs) in the following incorrect implementation of `check-children`.

Note: although this question is worth 5 points, there may be anywhere from 1 to 10 mistakes in this code.

```
(define (check-children tree)


  (or (= (first tree) (length (cdr tree)))


      (accumulate (lambda (x y) (and x y))


                  #f


                  (every check-children


                         (cdr tree) ))))
```

Question 12 (6 points):

Define the following stream. You may use helper procedures.

```
> (ss strm 15)
(1 1 2 1 2 3 1 2 3 4 1 2 3 4 5 ...)
```

$(\underline{1}\ \ \underline{1\ 2}\ \ \underline{1\ 2\ 3}\ \ \underline{1\ 2\ 3\ 4}\ \ \underline{1\ 2\ 3\ 4\ 5}\ \ \ldots)$

(Note: the underlines are just to clarify the problem for you; this is a single stream of numbers.)

Question 13 (7 points):

Consider the following object-oriented code for an `animal` class:

```
;; The animal class has a last-eaten instance variable
;; that always contains the thing that was last eaten.
(define-class (animal)
  (instance-vars (last-eaten #f))
  (method (eat something)
    (set! last-eaten something)))

(define bugs-bunny (instantiate animal))
(ask bugs-bunny 'eat 'carrot)
```

(a) We want to make a `fish` class. (A `fish` is a kind of `animal`.) Find and fix the problems in the `fish` class by adding any necessary code. There are at most four problems to fix.

```
(define-class (fish)



  (instance-vars (everything-eaten '())



  (method (eat something)



    (cons something everything-eaten) )



  (method (worms-eaten)



    (length (filter (lambda (thing-eaten) (equal? thing-eaten 'worm))



                    everything-eaten) )))
```

Question 13 continues on the next page.

Question 13 continued:

(b) A `goldfish` behaves exactly like a `fish`, but also has an `owner`. We are going to write a `goldfish` class.

Y ____ N ____ Should you make `fish` a parent of `goldfish`?

Y ____ N ____ Should you make `animal` a parent of `goldfish`?

Y ____ N ____ Should you include an instance variable (or instantiation variable) called `owner` in the `goldfish` class?

Y ____ N ____ Should you include an instance variable (or instantiation variable) called `last-eaten` in the `goldfish` class?

Y ____ N ____ Should you include an instance variable (or instantiation variable) called `everything-eaten` in the `goldfish` class?

Y ____ N ____ Should you write an `owner` method for the `goldfish` class?

Y ____ N ____ Should you write a new version of the `eat` method?

Y ____ N ____ Should you write a new version of the `worms-eaten` method?

Question 14 (6 points):

Write logic (query) language rules for `every-other`, a relation between two lists that is satisfied if and only if the second list is the same as the first list, but with every other element removed.

```
> (every-other (frodo merry sam pippin) ?x)
(every-other (frodo merry sam pippin) (frodo sam))
> (every-other (gandalf) ?x)
(every-other (gandalf) (gandalf))
```

Do not use `lisp-value`!