Instructions

- Question 0 (1 point) Fill in the front, write your name on the next page & comment on the exam on the right. Rate its difficulty, fairness, and add any other comments that come to mind.

- You have two hours to complete this midterm. The midterm is open book and open notes, no computers. Partial credit may be given for incomplete / wrong answers, so please write down as much of the solution as you can.

- You may not use any explicit recursion.

- You may NOT write auxiliary functions for a problem unless they are specifically allowed in the question. Feel free to use any Scheme function that was described in sections of the textbook we have read without defining it yourself, unless we specify otherwise.

- You do not need to write comments for functions you write unless you think the grader will not understand what you are trying to do otherwise.

- You may write the exam in pen or pencil and still be eligible for a regrade.

- You may write λ instead of lambda in your code.
Name:  

Question 1: He’s making a list...he’s checkin’ it twice... (10 pts, 15min)

- ONLY THIS QUESTION REQUIRES LIST FUNCTIONS.
- DO NOT USE LIST FUNCTIONS ANYWHERE ELSE ON THE EXAM.

Fill in the blanks. If you believe the return value is an error, write ERROR. (2 pts each)

a) (map length '( ' ) ) \rightarrow (2 1 0)

b) (c____ r ' ( (where) (is) (my) ) ) \rightarrow is

c) (append '(x) (cons '(y) '())) \rightarrow

d) (cons '(x) (list '(y) '())) \rightarrow

e) (list '(x) (append '(y) '())) \rightarrow

Question 2: I’m drawing a blank... (10 pts, 15min)

Fill in the blanks. Your answers must lie completely with the blanks (no extra auxiliary functions allowed).

a) Fill in the blank to define helper. You may not use if or cond or any hard-coded numbers (or expressions that evaluate to numbers) in your code. (3 points)

```
(define (helper __________ )
(define (foo pred? f n arg)
    (keep pred? ((repeated f n) arg)))

(foo odd? helper 2 '(1 2 3 4 5 6)) \rightarrow (3)
```

b) Rewrite bar using HOFs and NO EXPLICIT RECURSION. (4 points)

```
(define (bar fun pred? s)
    (cond ((empty? s) '())
        ((pred? (first s))
            (se (fun (first s)) (bar fun pred? (bf s))))
        (else
            (bar fun pred? (bf s))))))

(define (bar fun pred? s)

```

c) Complete the definition for mystery, which (trust us) IS POSSIBLE to write. (3 pts)

```
(define (mystery __________ )
    (every mystery '(you and you)) \rightarrow (you and me)
    (every mystery '(you me myself i)) \rightarrow ()
    (every mystery '(who are the coolest and smartest students?)) \rightarrow (you and me)
```

Question 3: How much knowledge have you accumulated? (10 pts, 30 min)

a) When teaching students what a HOF does, it's often useful to *expand* a call to a HOF into a simpler form. We're going to assume someone has written `expand`, which does exactly this for us (similar to what the *modeler* does). That is, it *first* shows the *expansion* to demonstrate what the HOF was *really doing*, then it evaluates and gives the return value. E.g.:

```
(expand (every square '(2 3)))  (se (square 2) (square 3))  (4 9)
(expand ((repeated square 2) 3))  (square (square 3))  81
```

...where "→" is the symbol for the *expansion* and “⇒” is the symbol for the return value. Show the expansion and return value for `(accumulate / '(', 9 6 2 1)):` (4 pts)

```
(expand (accumulate / '(', 9 6 2 1)))  ⇒
```

b) Given the new combiner *make-name*, which takes a *firstname* (e.g., gray) and a *lastname* (e.g., davis) and puts it into a single name (e.g., gray-davis), we want to write *get-firstname* (which gets back the *firstname* from name):

```
;; requires: firstname and lastname be non-empty and contain no "-"s
(define (make-name firstname lastname) (word firstname "-" lastname))

(make-name 'gray 'davis)  ⇒  gray-davis
(get-firstname 'gray-davis)  ⇒  gray
```

```
(define (get-firstname name)
  (accumulate get-firstname-helper name))

(define (get-firstname-helper arg1 arg2)
  (if (not (equal? (first arg2) "-")) ;; 1
    1
    (word arg1 arg2) ;; 2
    arg2)) ;; 3
```

*Sounds* easy, right? Shown above is our code for `get-firstname-helper` which has *exactly one buggy line*. Fill in the two sentences below. (6 points)

Calling `(get-firstname 'gray-davis)` returns ____________________

when it *should* return gray. Changing line ___ to ____________________

in `get-firstname-helper` fixes the problem so that `get-firstname` works perfectly.
Name:

**Question 4: Spiiiiiin the Wheel of Fortune!!** (10 pts, 30 min)

Write wheel-of-fortune using no explicit recursion and without using accumulate. Feel free to use a helper function called sentence->word whose domain is a sentence of letters and whose range is a word with all the letters smushed together. E.g., (sentence->word '(1 o v e)) \(\Rightarrow\) love (10 points)

```scheme
;; INPUTS : A phrase (sentence) and word of guessed letters
;; REQUIRES :
;; RETURNS : The original phrase with all the letters that have not been
;; guessed replaced with /-/. 
;; EXAMPLES : (wheel-of-fortune '(i love cs3) 'xy) \(\Rightarrow\) (- ---- ----)
; : (wheel-of-fortune '(i love cs3) 'xyoi3v) \(\Rightarrow\) (i -qv-- --3)
; : (wheel-of-fortune '(i love cs3) 'xyoi3vlesc) \(\Rightarrow\) (i love cs3)

(define (wheel-of-fortune phrase guessed) ;; NO EXPLICIT RECURSION!


```

**Question 5: March comes in like a lion and goes out like a fox (10 pts, 30 min)**

*(There are three parts here each worth 5 pts; we will throw out your lowest part)*

a) Remember the function swap-args from the first quiz? E.g., (swap-args 3 5) \(\Rightarrow\) 2

You realize you can now write new-swap-args which doesn't need to know the arguments in advance! Its domain is a binary function, and its range is the same binary function but with the arguments swapped. Fill in the blank only. (5 points)

```scheme
(define new-swap-args

(define minus-swapped (new-swap-args -))
(minus-swapped 3 5) \(\Rightarrow\) 2 ;; same as (- 5 3)

```

b) Remember all-good-partners you had to write for the online exam?

Using a call to all-good-partners and NO EXPLICIT RECURSION, write all-bad-partners. (which should return a sentence of all the partners that are NOT good; i.e., that all-good-partners would not return) (5 points)

```scheme
> (define (all-bad-partners good-partners? students)

(all-good-partners

(all-bad-partners same-name-length? (ana bo ohe dan ed))
(ana-bo ana-ed bo-che bo-dan che-ed dan-ed)

```

c) Fill in the blank so that the overall expression returns foo. (5 points)

```scheme
(lambda (y) (y y)) \(\Rightarrow\) foo

```