(define (recursion) (recursion))

Personal Information

<table>
<thead>
<tr>
<th>Last name</th>
<th>First Name</th>
<th>Student ID Number</th>
<th>Login</th>
<th>cs3-</th>
</tr>
</thead>
</table>

The name of your TA (please circle): Alex, Andrew, Anil, Lauren, Clint

Name of the person to your Left:

Name of the person to your Right:

All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS3 who have not taken it yet. (please sign)

Instructions

- You have two hours to complete this midterm. It is open book and open notes, no computers.
- Partial credit will be given for incomplete / wrong answers, so please write down as much of the solution as you can..
- Please turn off all pagers, cell phones and beepers. Remove all hats & headphones.
- Write the difficulty and fairness ratings in the boxes to the right and please add additional comments below.

Grading Results

<table>
<thead>
<tr>
<th>Question</th>
<th>Max. Points</th>
<th>Points Earned</th>
<th>Difficulty (0=easy 5=hard)</th>
<th>Fairness (0=fair 5=unfair)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please comment above & left:
**Question 1 : Blasts from the Past!! (9 points, 1 point each, we drop lowest)**

Assume the following procedures have already been defined:

```scheme
(define (infinite-loop) (infinite-loop))

(define (snack-or-not meal)
  ! (if (equal? meal (or 'breakfast 'lunch 'dinner))
      'not
      'snack))

(define (my-item num sw) ;; sw means sentence-or-word
  ! (if (equal? num 1)
      (first sw)
      (my-item (- num 1) (bf sw)))

(define (mood? people)
  ! (or (empty? people)
      (and (equal? (first people) 'happy)
           (mood? (bf people))))
```

Fill in the blanks below. If something is impossible, write **IMPOSSIBLE**. If something runs forever, write **RUNS-FOREVER**. If something will produce an error, write **ERROR**. You do not have to explain the error. The symbol `\(\Rightarrow\)` means “evaluates to”. The word **IMPOSSIBLE** can only appear in the input and **ERROR** can only appear in the output. For example,

\[
(+ 3 4) \Rightarrow 7
\]

1. \((\text{if \#f (infinite-loop) 'yankees}) \Rightarrow \text{__________}\)
2. \((\text{first (bf (last (bl '(cs3 is a great class))))}) \Rightarrow \text{__________}\)
3. \((\text{last \text{____________________________________}}) \Rightarrow \text{(recursion)}\)
4. \((\text{snack-or-not 'lunch}) \Rightarrow \text{__________}\)
5. \((\text{my-item 1.5 '(one two three four)}) \Rightarrow \text{__________}\)
6. \((\text{my-item \text{____________________} '(recursion)}) \Rightarrow \text{e}\)
7. \((\text{cond ('false 'yes!) (else 'no!))} \Rightarrow \text{__________}\)
8. \((\text{count (se (se 'fun) '()) 'recursion "" (word 'mid "" 'term))}) \Rightarrow \text{_______}\)
9. To what recursion pattern is \textit{mood?} closest? Circle one below:
   MAPPING    FINDING    FILTERING    COUNTING    TESTING    COMBINING
10. What is a better name for the \textit{mood?} predicate? Circle one below:
    all-unhappy? any-unhappy? all-happy? any-happy?
Question 2: 1,2,3,4,5! Wow! I have the same combo on my luggage! (10 points)

The function `combos-for` produces a sentence of ‘integer-integer’ combinations. It takes in a positive integer `n` as its only argument and returns a sentence of all the possible combinations of `n` and all numbers `1` through `n` separated by a `-`. E.g.,

```
(combos-for 1)  \rightarrow  (1-1)
(combos-for 3)  \rightarrow  (3-1 3-2 3-3)
```

Below is an attempt at `combos-for`:

```scheme
;; INPUT:   A positive integer
;; RETURNS: Returns a sentence of all combinations of n and 1-through-n

(define (combos-for n)
  (combos-for-helper n 1))

(define (combos-for-helper n i)
  (if (= n i)
      (word n '- i)
      (se (word n '- i)
        (combos-for-helper n (- i 1))))))
```

There are two bugs. Provide the answers to the following blanks:
(You may include ERROR or RUNS-FOREVER in your answers)

a) A call to `(combos-for 2)` results in __________________ when it should return
   (2-2 2-1) or (2-1 2-2). Replacing line ___ with _______________________
   fixes that particular bug.

b) After the fix, one bug still exists. Provide a valid call to `combos-for` that reveals
   the bug:`(combos-for ______ )  \rightarrow  ________________`. Replacing line ___ with
   ____________________________ fixes the final bug.
   After this fix, the function should perform as advertised on all valid input.

c) What type of recursion does the original `combos-for-helper` employ? Circle one:
   TAIL       EMBEDDED
Question 3: Spin that wheel, cut that pack, and roll those loaded dice! (10 pts)

We want the function `dice-combos` to return all the possible combinations of rolling two n-sided dice. The format for a combination is the same as in Question 2, where all combinations come in “integer-integer” format. Order does not matter within a word or sentence, so 1-2 and 2-1 mean the same thing, & (2-2 2-1 1-1) and (1-1 2-2 1-2) mean the same thing. However, i-j and j-i (where j and i are any numbers) should never both appear in the answer (see the examples below).

dice-combos takes in a positive integer n which represents the number of sides on one die and returns a sentence of all the combinations that can result from rolling two n-sided dice. Here are some examples:

(dice-combos 1) ➔ (1-1) ;; order doesn’t matter
(dice-combos 2) ➔ (2-2 2-1 1-1) ;; within word or sentence
(dice-combos 3) ➔ (3-3 3-2 3-1 2-2 2-1 1-1)

Fill in the blanks to complete the code below. Feel free to call any other procedures found on this test (called “software re-use”); assume they work as intended.

```
(define (dice-combos n)
  (dice-combos-helper _____________ _________________________________)

(define (dice-combos-helper n so-far)
  (if (= n 1)
      ____________________________________________________________________________________________
      (dice-combos-helper ______ __________________________)))
```

(i-was-bad 500)

NICE TRY.
Question 4: Putting the FUN back in function! (10 pts)
You’re given the following two functions:

```
(define (get-back w) ;; send the first letter to the back
  (word (bf w) (first w)))
```

```
(define (word-fun n w) ;; have some fun with words
  (if (= n 0)
      "" ;;
      (word w (word-fun (- n 1) (get-back w)))))
```

a) What does the call to (word-fun 1 'abc) return?

b) What does the call to (word-fun 3 'abc) return?

c) What would be a good name for the unnamed function below that takes in a number n and a letter L?

```
(define (_______________________ n L) (word-fun n L)) ;; L is a letter
```

d) What are the first and last three numbers of: (word-fun 1000000 1234567890)?

_________ …lots-of-numbers-in-the-middle… __________
Question 5: Two roads diverged in a wood... (10 pts)
You're lost in the forest. Every place in the forest is either a dead-end or has exactly 2 one-way paths: left and right. Your goal is to find out if there is a way home.

We introduce an abstract data type called a place, but you don’t know (and you don’t need to know) how it is represented; it could be a word, sentence, boolean, or number. You are presented with four operations (all take a place as an argument):

(home? place) returns #t if the place is your home, #f otherwise.
(dead-end? place) returns #t if the place is a dead-end (i.e., no paths from it).
(go-left place) follows the left path, returning a new place.
(go-right place) follows the right path, returning a new place.

It is an error to go-left or go-right from a dead-end (because it has no paths!). There is no way in this forest to follow a sequence of left paths and/or right paths and end up where you started. I.e., there’s no way to walk in circles. Your home (if one exists) might be at a dead-end or it might not. You might actually start your search at home.

Write (path-home? place) which uses the four functions above and returns #t if you can get home following a (possibly zero) number of lefts & rights starting from place and #f otherwise.

```
(define (path-home? place)

_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________
_______________________________________________________________________

(home? ) ➔ #f
(home? ) ➔ #f
(home? ) ➔ #f
(home? ) ➔ #f
(home? ) ➔ #f

(dead-end? ) ➔ #f
(dead-end? ) ➔ #f
(dead-end? ) ➔ #f
(dead-end? ) ➔ #f

(go-left ) ➔ (path-home? ) ➔ #t
(go-right ) ➔ (path-home? ) ➔ #t
(go-right ) ➔ (path-home? ) ➔ #f

;; go-left or go-right from , or [] is an ERROR
```

Example:

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I–
I took the one less traveled by,
And that has made all the difference.

– Robert Frost