CS 3 Midterm 1 Review

1. **Quick Evaluations** – Indicate what each of the following would return if typed into STK. If you think it would error, then please write ERROR. If you think that it would loop forever, write LOOPS.

   (count (sentence ‘I ‘am ‘(sentence) ‘at ‘a ‘cs3 ‘review)) → **7**

   (count (word ‘I ‘am ‘(sentence) ‘at ‘a ‘cs3 ‘review)) → **ERROR**

   ```
   (define (mystery wd)
     (or   (and  (vowel? (first wd))
                 (mystery (bf wd)))
           (empty? wd)))
   ```

   (mystery ‘aeiou) → **ERROR**

   (appearances ‘at ‘atbtatbtat) → **0**

   ```
   (equal?  (or  ‘this ‘cannot ‘be ‘right)
            (and ‘neither ‘can ‘this))      → #t
   ```

   (member? ‘bat (sentence (word ‘bat ‘mouse) (sentence ‘bat ‘mouse))) → **#t**

(item 4 '(a b c d e f g)) → **d**

```
(define (both-even? sent)
    (and   (even? (first sent))
           (even? (bf sent))))
```

(both-even? '(2 4)) → **ERROR**

2. **More Celebrity Questions** – Recall the celebrity quiz from lab. Here is the definition, incase you have forgotten:

Assume that you are working with a "celebrity" program that someone else wrote. This package gives you accessors (or selectors) to work with a "celebrity". You won't change (or even look at) those procedures, but will use them. Some of these accessors include

* name, which takes a single celebrity and returns the name as a single word. For example, if you have stored a celebrity in the variable cel,

```
(name cel)
```

might return the word Joe.

* hair-color which takes a celebrity and returns one of the words blond, brown, black, or red
* the procedure movies, which returns a sentence of movies that the celebrity has been in
* and many other procedures

A variable that "is" a celebrity is a number that, by itself, doesn't contain any of the information (like name, etc). For instance, cel above might simply be the number 5. Only by using the accessors can the information about a celebrity be pulled out.

One of the accessors written is *height*. It returns the height of the celebrity, in inches. For example, given a certain cel, we might find that

(height cel) → 68

Write a procedure average-height which takes a sentence of celebrities and returns a single number: the average height of the celebrities in the sentence. Remember, average is computed as the sum divided by total number. It is ok to use a helper procedure

```
(define  (average-height celebrities-sent)
    (/ (total-height celebrities-sent) (count celebrities-sent) ))

(define  (total-height celebrities-sent)
    (if (empty? celebrities-sent)
        0
        (+ (height (first celebrities-sent)) (total-height (bf celebrities-sent)) )))
```

3. **Valentines Day Matchmaking** – We have come up with a brilliant new method of deciding if people would be a good match for each other. We simply need four pieces of information, three of which is stored in a sentence, and those things are favorite movie, favorite sport, and favorite color. The favorite sport is stored as a single word. The favorite color is the second word of the sentence. The movie is the rest. The final piece of information is the age of each person, stored simply as a number. We need a simple predicate procedure that, given six sentences, returns true if these two people would be a good match for each other, false otherwise. A good match is defined simply as if exactly two of the three subjects are equal. If all three match, they are too similar and it won't work. If only one or none match, then they don't have enough in common and again it won't work. In any case where a good match has been found, we must make sure that the two ages are within 3 years of each other. You may find the (abs) function useful, which takes two numbers and returns the absolute value.

   An example call would be (good-match? '(soccer blue top gun) '(football green top gun) 21 23) → #f

```
(define (good-match? subjects1 subjects2 age1 age2)
        (cond
                ((> (abs (- age1 age2) 3) #f)
                ((and (equal? (movie subjects1)  (movie subjects2))
                      (equal? (sport subjects1) (sport subjects2))
                      (equal? (color subjects1) (color subjects2)))      #f)

                ((equal? (movie subjects1)  (movie subjects2))
                        (or (equal? (sport subjects1) (sport subjects2))
                            (equal? (color subjects1) (color subjects2))))

                ((equal? (sport subjects1) (sport subjects2))
                        (equal? (color subjects1) (color subjects2)))
```

```
            (else #f)
        ))

(define (movie subject)
        (bf (bf subject)))
(define (sport subject)
        (first subject))
(define (color subject)
        (first (bf subject)))
```

4. **Debugging a Biologists' Work –** A Biologist friend of yours wrote a procedure to check if two strands of DNA of the same length have more than 2 differences in their Genetic Code. Recall that DNA can be written out as a sequence of A T G and C. (Don't worry, you don't have to remember what this means). Thus, strands with two differences might be (A T G C) (A T A T). These two strands would be considered similar. The following code was written, taking each strand as a sentence. The strands are guaranteed not to be empty when starting.

   a) Attempt 1 -

```
(define (similar-strands?  strand1 strand2)
        (and (equal? (count strand1) (count strand2))
             (empty? strand1)
             (and    (equal? (first strand1) (first strand2))
                     (similar-strands? (bf strand1) (bf strand2)))))
```

   The Biologist knows that this procedure doesn't allow for one error in the strands, however, whenever the biologist enters ANY strands, the program returns #f. What is wrong with this procedure?

   The empty? check fails because of the AND.
   We don't need to write the Correct Code since the Problem Didn't Ask for it. But, it is important to note that using and/or is useful for recursion.

   b) Attempt 2 – After helping

```
(define (similar-strands? strand1 strand2)
       (if (not (equal? (count strand1) (count strand2)))
         #f
         (similar-strands-helper strand1 strand2 0)))
```

```
(define (similar-strands-helper strand1 strand2 num-errors)
        (cond
1.              ((empty? strand1) #t)
2.              ((> num-errors 2) #f)
3.              ((equal? (first strand1) (first strand2))
                    (similar-strands-helper strand1 strand2 num-errors))
4.              (else
                    (similar-strands-helper (bf strand1) (bf strand2) 1))))
```

For each Line – Tell whether it is a Base Case or Recursive Case, The correction to the Code if it needs to be fixed, otherwise NOTHING, and What line it should be swapped with, if Any, otherwise write NONE.

Base or Recursive?	New Code	Swap with line?
1. **BASE**	**NOTHING**	**2**
2. **BASE**	**NOTHING**	**1**
3. **RECURSIVE**	**(similar-strands-helper (bf strand1) (bf strand2) num-errors)**	**NONE**
4. **RECURSIVE**	**(similar-strands-helper (bf strand1) (bf strand2) (+ 1 num-errors))**	**NONE**

5. The Truth about Sports – We are writing a sports survey program for a company, but we are a little biased. The company asks, in its survey, for a person to give, in a sentence, his/her three favorite sports. There are four possible options for these sentences. Either they have three sports listed, as in

---

(football is my favorite followed by hockey and tennis)

in the precise format of (* sport* is my favorite followed by *sport * and *sport *)

---

or else, they only have one overwhelming favorite and give

(basketball is my favorite in all the world and I love it and nothing else)
(basketball is my favorite and nothing else)

in the precise format of (* sport* is my favorite *extra nonsense words* and nothing else).
(namely, we don't know how long this sentence might be).

---

the third options is they may not like any sport much
(well I do not really like anything but I would have to pick basketball)
(well I do not really like much but I suppose basketball)

in the precise format of (well I do *extra nonsense words* *sport *)
---

and finally, if they despise all sports, then they would respond with
(well I hate sports but if I had to pick I would say cricket possibly)
(well I hate them all but I guess tennis maybe)

in the precise format of (well I hate *extra nonsense words * * sport* *extra word*)

We want to write a survey program that, given such a sentence, replaces all the instances of any of the sports with the sport soccer. We do NOT want to get rid of any of the other words in the sentence or our boss will know we've tampered with something.

Call this procedure ***survey-correction***. Assume first, second, and third are all defined.

```
(define (survey-correection sports-survey)
        (cond
                ((equal? (se (first sports-survey)
                             (second sports-survey)
                             (third sports-survey)) '(well I hate))
                         (se (bl (bl sports-survey)) 'soccer (last sports-survey)))
                ((equal? (se (first (sports-survey)
                             (second sports-survey)
                             (third sports-survey)) '(well I do))
                         (se (bl sports-survey)) 'soccer (last sports-survey)))
                ((equal? (last sent) 'else)
                         (se 'soccer (bf sports-survey)))
                (else
                         '(soccer is my favorite followed by soccer and soccer))))
```