

CS3 Sample Final Solutions

Problem 1

List the different ways that the `position` argument to `ttt` can be illegal. (Two ways count as different if they would be checked with different code.)

```
length 9, all chars either x or o or _

(define (legal? board)
  (and
    (= (count board) 9)
    (equal? board
      (keep (lambda (char) (member? char 'xo_)) board)) ) )

the number of x's is equal to the number of o's and there is no x
win, or the number of x's is one less than the number of o's and
not an o win

(and
  ...
  (or
    (and
      (= (appearances 'x board) (appearances 'o board))
      (not (win? 'x board)) )
    (and
      (= (appearances 'x board) (+ 1 (appearances 'o board)))
      (not (win? 'o board)) )
```

Problem 2

Define a Scheme procedure named `updated-hands` that takes two arguments. The first is a list of two hands, each a sentence of cards, with each card being a word in which the first letter is the suit and the rest is the rank. The second is a rank. `Updated-hands` should return the result—again, a list of two hands—of transferring a card of the given rank from the second hand to the first. You may assume that the second hand contains at least one card of the given rank, and that hands do not contain duplicate cards. Here's an example:

expression	desired result
<pre>(updated-hands '((ha hk c2) (d5 c6 h10 d10 c8)) 10)</pre>	<pre>((h10 ha hk c2) (d5 c6 d10 c8)) or ((d10 ha hk c2) (d5 c6 h10 c8))</pre>

```

(define (updated-hands 2hands rank)
  (let ((card (find (cadr 2hands) rank)))
    (list
     (se card (car 2hands))
     (remove-once card (cadr 2hands)) ) ) )

(define (find hand rank)
  (first (keep (lambda (card) (equal? (bf card) rank))
             hand)) )

```

(remove-once appears on page 236 of [Simply Scheme](#))

Problem 3

Consider a data base for a programming class, represented as a list, each element of which has the form

```
(student-name project partner-list)
```

There is one element in the data base list for each student in the class. The element contains the student's name, the name of the project she worked on, and a list of the other students in the class with whom she worked in partnership on the project. This list does not include the student herself.

For this problem you are to write two versions of a procedure `num-submissions` that, given a data base and the name of a project, returns the number of submissions of that project. One project per partnership will be submitted, so each person in a partnership of size N will submit $1/N$ projects. Partnerships in this class may be of any size (unlike in CS3 this semester).

Part a

Write a version of `num-submissions` that does not use higher-order procedures. (It may use recursion.) You may use helper procedures, as long as they too do not use higher-order procedures.

```

(define (num-submissions db proj)
  (if (null? db) 0
      (+ (contrib proj (car db)) (num proj (cdr db))) ) )

(define (contrib proj record)
  (if (equal? (project record) proj)
      (/ 1 (+ 1 (length (partners record))))
      0) )

```

Part b

Write a version of `num-submissions` that does not use recursion. (It may use higher-order procedures.) You may use helper procedures, as long as their definitions do not use recursion.

```
(define (num-submissions db proj)
  (reduce +          ;; apply or accumulate is ok
    (map
      (lambda (record) (/ 1 (+ 1 (length (partners record)))))
      (filter
        (lambda (record) (equal? (project record) proj))
        db) ) ) )
```

Problem 4

Given below is a version of the `flatten` procedure from one of the lab activities. It doesn't work correctly.

```
; L is a list. Return the list of the atoms that occur in L,
; in the left-to-right order they appear in L.
(define (flatten L)
  (if (null? L) '()
      (reduce append (map flatten L)) ) )
```

Part a

By filling in the blank below, provide a nonempty argument to `flatten` that does *not* cause an error when the resulting expression is evaluated.

(flatten _____)

|| A list in which all elements are empty lists

Part b

List the steps of the evaluation of `(flatten '(A B))`. (That is, imitate the Replacement Modeler.) If the result is an error, describe the error message and give the expression whose evaluation produced the error.

```
|| ERROR: input to map isn't a list:
   (map flatten 'A) or (map flatten 'B)
```

Part c

Which of the following is a reason that `flatten` doesn't work correctly? (More than one may apply.)

1. The base case test `(null? L)` is incorrect.
2. The base case test is correct, but the empty list is the wrong result to return in that case.
3. The input passed to the recursive call is not what the procedure expects.
4. The input passed to the recursive call doesn't get closer to the base case.
5. `Reduce` and `append` are the wrong procedures to use with `(map flatten L)` to get the answer.

Explain your answer(s).

- 1: The base case should be testing for `(word? L)` and returning `(list L)`.
- 3: The procedure wants a list and is getting a word.

Part d

Making as few changes as possible to `flatten`, fix it. Here's a copy of the code; indicate clearly what should be inserted, what should be deleted, and what should be changed.

```
(define (flatten L)
  (if (null? L) '()
      (reduce append (map flatten L)) ) )
```

```
(define (flatten L)
  (if (word? L) (list L)
      (reduce append (map flatten L)) ) )
```

Problem 5

Consider the following higher-order procedure, which returns the result of combining some of the elements in its list argument into lists.

```
(define (some-paired L pred?)
  (cond
    ((null? L) L) ; case 1
    ((null? (cdr L)) L) ; case 2
    ((pred? (car L) (cadr L)) ; case 3; #t means don't pair
     (cons (car L) (some-paired (cdr L) pred?)))
    (else ; case 4
     (cons
      (list (car L) (cadr L))
      (some-paired (caddr L) pred?) ) ) ) )
```

Part a

One may use `some-paired` to write a procedure `grouped-into-frames` that groups balls in a bowling game into frames, leaving strikes ungrouped. `Grouped-into-frames`, given the list

```
(9 1 0 10 10 10 6 2 7 3 8 2 10 9 0 2 7)
```

would thus return the list

```
((9 1) (0 10) 10 10 (6 2) (7 3) (8 2) 10 (9 0) (2 7))
```

Fill in the blank below with a procedure that completes the definition of `grouped-into-frames` in terms of `some-paired`. Don't add anything else to the `grouped-into-frames` procedure; you may, however, supply a helper procedure. Assume for this part that the argument to `grouped-into-frames` is a list representing a game with a miss in the tenth frame.

```
(define (grouped-into-frames ball-list)
  (some-paired
    ball-list
    
      (lambda (ball11 ball12) (= ball11 10))
    
  ) )
```

Part b (graded only if part a is essentially correct)

What would be the result of supplying the ball list below to your procedure from part a? (Note the strikes in the tenth frame.) Briefly explain your answer by identifying the cases in `some-paired` that handle the last three balls.

```
(9 1 0 10 10 10 6 2 7 3 8 2 10 9 0 10 10 8)
```

```
((9 1) (0 10) 10 10 (6 2) (7 3) (8 2) 10 (9 0) 10 10 8)
```

The strikes in the tenth frame are handled in case 3 and left untouched by `some-paired`; then the `(null? (cdr L))` in case 2 succeeds.

Problem 6

Consider a procedure named `1-extra?` that, given two words as inputs, should return `true` exactly when the first word is the result of inserting a single letter into the second word. Examples:

expression	desired result
<code>(1-extra? 'HEAT 'HAT)</code>	<code>true</code>
<code>(1-extra? 'THAT 'HAT)</code>	<code>true</code>
<code>(1-extra? 'HATE 'HAT)</code>	<code>true</code>
<code>(1-extra? 'ABBC 'ABC)</code>	<code>true</code>
<code>(1-extra? 'AT 'HAT)</code>	<code>#f</code>
<code>(1-extra? 'HAT 'HAT)</code>	<code>#f</code>
<code>(1-extra? 'HATED 'HAT)</code>	<code>#f</code>
<code>(1-extra? 'CHEAT 'HAT)</code>	<code>#f</code>

A buggy version of `1-extra?` appears below.

```
(define (1extra? big small)
  (= (num-matches big small) (- (count big) 1)) )

(define (num-matches big small)
  (cond
    ((empty? small) 0)
    ((equal? (first big) (first small))
     (+ 1 (num-matches (bf big) (bf small))))
    (else (num-matches (bf big) small)) ) )
```

Part a

Provide a call to `1-extra?` for which it does not perform as intended, that is, it returns `true` or crashes when `#f` is the correct answer, or it returns `#f` or crashes when `true` is the correct answer.

```
|| (1-extra? 'AT 'HAT)
```

Part b

Describe, as completely as possible, the set of words `wd` for which evaluating the expression

```
(1-extra? wd 'HAT)
```

produces the correct answer. Briefly explain your answer.

Whenever `wd` contains the letters `H`, `A`, and `T` in order, it returns the correct answer. Otherwise it crashes. That's because both recursive calls to `num-matches` decrease the length of `big`, but the base case doesn't check whether `big` is empty.

Problem 7

Consider the following procedure, intended to return the result of inserting commas in the correct places (every third position starting from the right) in its nonnegative integer argument.

```
(define (with-commas n)
  (if (< n 1000)
      n
      (word (with-commas (quotient n 1000)) "," (remainder n 1000)) ) )
```

For example, `(with-commas 12345678)` should return the word `12,345,678`. The procedure above, however, has a bug.

Part a

Describe, as completely as possible, the set of nonnegative integer arguments for which `with-commas` does not work correctly.

Any value in which a 0 would follow a comma.

Part b

Fix the bug, changing the `with-commas` procedure as little as possible. You may define a helper procedure.

Replace the call to `remainder` by `(with-leading-zeroes (remainder n 1000))`:

```
(define (with-leading-zeroes lt1000)
  (cond
    ((< lt1000 10) (word "00" lt1000))
    ((< lt1000 100) (word "0" lt1000))
    (else lt1000) ) )
```