CS116 Winter 2011 Assignment 1
Due: Tuesday, January 18 at 10:00AM

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Important Information:

1. Read the course Style Guide for information on assignment policies and how to organize and submit your work. In particular, your solutions should be placed in files `a1qY.rkt`, where `Y` is a value from 1 to 5.
2. Be sure to download the interface file to get started.
3. Do not copy the purpose directly from the assignment description – it should be in your own words.
4. You should use abstract list functions (`map, filter, foldr`) where appropriate. **Solutions that use explicit recursion will not receive any correctness marks**.
5. All helper functions should be contained within the main function definition using `local`.
6. Do not use `reverse`, `build-list`, or `lambda`.
7. You may assume that all consumed data satisfies any stated assumptions, unless indicated otherwise.

Language level: Intermediate Student.
Coverage: Module 1

The following structures are needed for this assignment.

```
(define-struct clock (hour min))
;; A clock is a structure (make-clock h m), where
;; h is an integer between 0 and 23 (the hour of the day, 0
;;   is midnight)
;; m is an integer between 0 and 59 (the minute after the
;;   hour h)

(define-struct appointment (title start end))
;; An appointment is a structure (make-appointment t s e),
;; where,
;; t is a string (for the title of the appointment)
;; s is a clock (for the starting time of an appointment),
;; e is a clock (for the ending time of an appointment)
;; and where s, e refer to times on the same day,
;; and s occurs before e.

(define-struct tweet (sender message))
;; A tweet is a structure (make-tweet s m) where
;; s is a string for the sender's name,
;; m is a string for the sender's message (maximum length
;; 140 characters).
```

```
;; A gradelist is empty, or (cons g gl), where
;; g is a nat between 0 and 100, and
;; gl is a gradelist.

(define-struct course-result (title grades))
;; A course-result is a structure (make-course-result t g),
;; where
;; t is a string (title of course),
;; g is a gradelist (grades of all students in the course)
```

1. Complete the Scheme function `count-multiples`, which consumes a list of integers, and a single natural number `n`, and produces the number of values in the list which are multiples of `n`. For example,
   `(count-multiples (list 20 27 -10 2 11 0) 10)` => 3.

2. Complete the Scheme function `alter-string`, which consumes a string and produces a new string like the original, except that all non-alphabetical characters are replaced with a single space. For example,
   `(alter-string "Happy,happy!")` => `"Happy happy "`.
   The built-in character function `char-alphabetic?`, and the `string->list` and `list->string` conversion functions will be very useful in your solution.

3. Complete the Scheme function `cancel-long-appointments`, which consumes a list of `appointment` structures, and a natural number `too-long`, and produces a list containing only those `appointments` which are scheduled to last no longer than `too-long` minutes. For example,

```
(cancel-long-appointments
  (list (make-appointment "course meeting"
                  (make-clock 9 0) (make-clock 10 0))
        (make-appointment "lunch"
                  (make-clock 12 0) (make-clock 14 0)))
   60)
=> (list (make-appointment "course meeting"
                  (make-clock 9 0) (make-clock 10 0))).
```

4. Complete the Scheme function `average-tweet-length`, which consumes a list of `tweet` structures, and a string `id`, and produces the average length (in characters) for all the `tweets` in the consumed list that were sent by `id`. If there were no tweets sent by `id`, then the function should produce the symbol `'no-tweets`. For example,

```
(average-tweet-length
 (list (make-tweet "lmc" "reading")
       (make-tweet "jp" "making supper")
       (make-tweet "lmc" "eating")) "lmc") => 6.5
```

5. Complete the Scheme function `most-As` that consumes a list of `course-result` structures and produces a list containing the names of all the courses which have the greatest number of A grades (i.e. grades 80 or higher). For example,

```
(most-As
  (list
    (make-course-result "CS115" (list 89 72 45 96 80))
    (make-course-result "CS135" (list 90 37 90))
    (make-course-result "CS100" (list 70 80 90 85 40))))
=> (list "CS115" "CS100"))
```