CS 135 Fall 2015
Akinyemi, Heinle, Holtby, Istead, Roegiest, Tompkins

Assignment: 8

Due:	Tuesday, November 17, 2015 9:00pm
Language level:	Intermediate Student
Allowed recursion:	Pure Structural and Structural Recursion with Accumulators
Files to submit:	`email.rkt`, `offenders.rkt`, `abstract.rkt`
Warmup exercises:	HtDP 17.3.1, 17.6.4, 19.1.5, 20.1.1, 20.1.2, 24.0.7, 24.0.8
Practise exercises:	HtDP 17.6.5, 17.6.6, 19.1.6, 20.1.3, 21.2.3, 24.0.9

- All helper functions must be **local** definitions.
- You are not required to provide examples or tests for **local** function definitions. You are still encouraged to do informal testing of your helper functions outside of your main function to ensure they are working properly. Functions defined at the "top level" must include the complete design recipe.
- You may define global constants if they are used for more than one part of a question. This includes defining constants for examples and tests. Constants that are only used by one top level function should be included in the **local** definitions.
- In this assignment your *Code Complexity/Quality* grade will be determined both by how clear your approach to solving the problem is, and how effectively you use **local** constant and function definitions. You should include **local** definitions to avoid repetition of common subexpressions, to improve readability of expressions and to improve efficiency of code.
- You may only use the list functions that have been discussed in the notes up to the end of Module 10, unless explicitly allowed in the question.
- You may reuse the provided examples, but you should ensure you have an appropriate number of examples and tests.
- Your solutions must be entirely your own work.
- Solutions will be marked for both correctness and good style as outlined in the Style Guide.

Here are the assignment questions you need to submit.

1. Perform the Assignment 8 questions using the online evaluation "Stepping Problems" tool linked to the course web page and available at

   https://www.student.cs.uwaterloo.ca/~cs135/stepping/

   The instructions are the same as A03 and A04; check there for more information, if necessary.

   Reminder: You should not use DrRacket's Stepper to help you with this question, for a few reasons. First, as mentioned in class, DrRacket's evaluation rules are slightly different from the ones presented in class; you are to use the ones presented in class. Second, in an exam situation, of course, you will not have the Stepper to help you. Third, you can re-enter steps as many times as necessary to get them correct, so you might as well maximize the educational benefit.

2. Purple Chicken Tech Company (PCTC) wants you to help them analyze their email database. To start with, you will be examining aggregate data only. For each day an employee works, the database contains an *Email-Record*, a structure with 3 fields: the day-id (a natural number, with day 0 representing the day PCTC was founded), the number of hours worked that day, and the number of emails the employee sent that day.

   These records are tied to an employee using a *Daily-Stats* structure. This structure also has 3 fields: the ID number of the employee, the name of the employee, and a list of their daily email records. Data and structure definitions for these records can be found below, along with some sample data.

   ```
   (define-struct email-record (day-id hours-worked emails-sent))
   ;; An Email-Record is a (make-email-record Nat Num Nat)
   ;; requires: hours-worked >= 0

   (define-struct daily-stats (staff-id staff-name emails))
   ;; A Daily-Stats is a (make-daily-stats Nat Str (listof Email-Record))
   ;; requires: each day-id in emails is unique

   (define stats1 (list (make-email-record 1 7 20) (make-email-record 2 8 50)
                        (make-email-record 3 6 30) (make-email-record 4 8 100)
                        (make-email-record 5 7 50)))
   (define em1 (make-daily-stats 1 "Justin" empty))
   (define em2 (make-daily-stats 2 "Tolu" stats1))
   ```

   (a) Write a function called *avg-emails* that consumes a *Daily-Stats* and produces the average emails sent per hour by the staff member in *Daily-Stats*. If the *emails*, a (listof *Email-Record*) in *Daily-Stats* is empty, the function should produce 0. For example,

   ```
   (check-expect (avg-emails em1) 0)
   (check-expect (avg-emails em2) 250/36)
   ```

All helper functions must be encapsulated inside a **local** block in your main function.

(b) Write a function called *highest-email-record* that consumes a *Daily-Stats* and produces a (listof *Email-Record*) with the largest number of emails sent.

```
(define stats2 (list (make-email-record 1 10 80) (make-email-record 2 5 30)
                     (make-email-record 3 7 30) (make-email-record 4 7 80)
                     (make-email-record 5 12 80)))
(define em3 (make-daily-stats 10 "Liz" stats2))

(highest-email-record em3) ⇒ (list (make-email-record 1 10 80)
                                   (make-email-record 4 7 80)
                                   (make-email-record 5 12 80))
(highest-email-record em2) ⇒ (list (make-email-record 4 8 100))
```

All helper functions must be encapsulated inside a **local** block in your main function.

Place your functions in the `email.rkt` file.

3. At PCTC, staff members send a non-trivial amount of emails. The CEO wants to understand the impact of email-related activities on the productivity and profitability of the company. As a result, the CEO asked you to provide data analytics on their email database.

   For this task, you are going to analyze emails using a different structure than the previous one. An email has a sender represented by the staff-id (a *Nat*), recipients represented by a list of ids of staff members who received an email (a (listof *Nat*)), a word-count that denotes the number of words in the email (a *Nat*), and a reply (a *Reply*). A reply is a list of all emails sent directly in response to the original email. E.g. if there are emails A, B, and C, where B is a reply to A, and C is a reply to B, then A's list of replies would only contain B, not C. Only B's list of replies would contain C. Additionally, an email can only be the reply to a single other email. Phrased differently, each email is a node in a general tree, and its children are its direct replies.

```
;; A Reply is a (listof Email)

(define email7 (make-email 1 (list 2 3) 5 empty))
(define email6 (make-email 2 (list 1 4 5) 10 (list email7)))
(define email5 (make-email 3 (list 6 7) 15 (list email6)))
(define email4 (make-email 1 (list 2 3 4 5 6 7) 20 empty))
(define email3 (make-email 2 (list 3 5 6) 25 empty))
(define email2 (make-email 3 (list 4 7) 30 empty))
(define email1 (make-email 1 (list 2 3 4 5 6 7) 35 (list email2 email3 email4)))
```

A note about email counting: An email sent to many recipients is counted as one email. For example, if a person sends an email to 10 recipients, it will be counted as one (not 10) email.

(a) Define a structure called *email* that encapsulates the data above and provide a data definition. If there are requirements on the data, include them in your data definition.

(b) Write a function called *total-word-count* that consumes an *Email* and produces the total number of words in the *Email* including all words in the reply to the *Email*.

(*total-word-count email1*) ⇒ 110
(*total-word-count email6*) ⇒ 15

All helper functions you create must be encapsulated in a **local** block inside the main function.

(c) Write a function called *unique-email-senders* that consumes a (listof *Email*) and produces a list of all unique email senders in the consumed (listof *Email*). Unique email senders include all staff members who sent a reply to an email, but do not count the same person more than once. All helper functions must be encapsulated inside a **local** block in your main function. For example:

(*unique-email-senders* (*list email2 email3*)) ⇒ (*list* 2 3)
(*unique-email-senders* (*list email6*)) ⇒ (*list* 1 2)

(d) Write a function called *sent-email-summary* that consumes a (listof *Email*) and produces a list of all unique email senders in the consumed (listof *Email*) with the number of emails they sent. You may use functions you created for a previous part of the question, but any additional functions you write (helper functions) must be encapsulated inside a **local** block in your main function. For example:

(*sent-email-summary* (*list email1*)) ⇒ (*list* (*list* 1 2) (*list* 2 1) (*list* 3 1))

(e) It was rumoured that some staff members abuse the email service at PCTC. They have been sending numerous emails that impacted the general performance of the email server. The HR Manager is eager to know the email abusers as potential candidates to fire during the next phase of the restructuring process.

Write a function called *email-offenders* that consumes a (listof *Email*) and a threshold amount (a *Nat*) and produces a list of all staff ids who have sent more emails than the specified threshold. The list should be sorted in a non-decreasing order. You can use the Racket function *sort*.

You may use functions you created for a previous part of the question, but any additional functions you write (helper functions) must be encapsulated inside a **local** block in your main function. For example,

(*email-offenders* (*list email1*) 1) ⇒ (*list* 1)
(*email-offenders* (*list email1*) 3) ⇒ *empty*

Submit your code in a file named `offenders.rkt`.

Due to lectures not yet covering all of the required abstract list functions, the following question has now been made a 5% bonus. You are not required to hand in this question, but we encourage you to look ahead in the notes and attempt it.

4. Write a function *avg-without-recursion* that consumes a list of natural numbers and produces a list of three numbers as follows: (i) average of the entire list, (ii) average of only the even numbers, and (iii) average of only the odd numbers in the list. The list must contain at least one even and one odd natural number. For example,

   (*avg-without-recursion* (*list* 5 2 3 5 6 7 9 25)) ⇒ (*list* 7.75 4 9)

   For this question only:

   - You are required to use abstract list functions and you may only use *filter*, *map*, and *foldr*.
   - You are **not allowed to use explicit recursion**.
   - You **may not define any helper function**.

Place your function in the file `abstract.rkt`.

This concludes the list of questions for which you need to submit solutions. As always, do not forget to check your email for the basic test results after making a submission.
