CS 1101 A-Term 2021 Professor Engling

Homework 6 - Programming with Mutable Variables; Accumulator-Style Recursion

(140 Points -- 20 Bonus Points Available)

Due: Tuesday, October 12 at 11pm

Read the Expectations on Homework posted on Canvas.

Assignment Goals

- To make sure you can create linked data structures and write programs using mutable variables
- To make sure you can write programs in accumulator-style

The Assignment: A Simple Email System

In a simple email system, **user**s have **username**s and a collection of **message**s in their **mailbox**. A **message** consists of the sender's **username**, the **text** of the message (a String), and a flag indicating whether or not the user has **read?** the message. An email-system is a collection of users (**ListOfUser**).

A Word about Testing

As we will discuss in class, testing programs that use mutable variables becomes more complicated than testing programs that don't use mutable variables.
If you write helper functions for this assignment that can be tested with check-expect, you should provide test cases for those functions, as usual.
You do not have to show test cases that result in error conditions.

You do not have to show test cases for any function that involves mutation. The graders will be running your programs with their own set of tests (see below). However, make sure you understand how to construct tests for functions that use mutation (you might be asked something about this on the third exam).

Problems

1. **(15 Points)** Write the data definitions for an email-system (**ListOfUser**). Provide data definitions for a **message** and a **user**. As mentioned above, each **message** consists of the sender's **username** (a String), the **text** of the message (a String), and a (Boolean) flag indicating whether or not the (recipient) user has **read?** the message; and a **user** consists of a **username** (a String) and a **mailbox** (a ListOfMessage).

2. **(10 Points)** Create and document two variables: variable **`mailsys`**, an **`empty`** email system (with no users) and variable **`newuser`**, a user with name "Newuser" and an **`empty`** list of messages.

3. **(15 Points / 5 Bonus)** Write a function `add-user` that consumes a username and produces void. The effect of the function is to add a new user with the given username to the mail system. The new user should have an empty mailbox. You may assume that the username is not already in the mail system.

4. **(15 Points / 5 Bonus)** Write a function `send-email` that consumes the name of the sender of an email, the name of the recipient of the email, and the text of an email message, and produces void. The effect of the function is to store a new unread message in the recipient's mailbox. Assume the named recipient is a user in the mail system.

5. **(20 Points)** Write a function `get-unread-messages` that consumes a username and produces a list of messages. The produced list contains the unread messages in the mailbox of the user with the given name. Assume the username is a valid user in the mail system. An effect of the function is that all unread messages in the named user's mailbox have been set to read.

6. **(25 Points)** Write a function `most-messages` that doesn't consume anything. The function produces the user in the mailsystem with the largest number of messages in his/her mailbox. If there are no users in the system, the function produces an appropriate error. If two or more users have the most messages, the function just needs to return one of them (it doesn't matter which one). You must use accumulator-style programming to solve this problem.

7. **(10 Bonus Points)** Show a sequence of test cases that you would use to test the email system. [One point for each -- max three for each of four functions above.]

The graders will be running your programs with a set of test cases such as the following (there will be additional tests besides the ones shown):

```
(add-user "Gompei")
(add-user "Attila")
(send-email "Gompei" "Attila" "Goat big or goat home!")
(send-email "Attila" "Gompei" "You must be quackers!")
(send-email "Gompei" "Attila" "You have goat to be
kidding!")
(get-unread-messages "Newuser")
(get-unread-messages "Gompei")
(get-unread-messages "Attila")
(most-messages)
```

Make sure that your functions adhere to the signature/purpose given, otherwise the test cases won't run. Also, make sure you name the functions as indicated in each problem.

Accumulator-Style Recursion

8. **(20 Points)** Using accumulator-style recursion, write a function `total-string-length` that consumes a `ListOfString` and produces the sum of the lengths of the strings in the list.

9. **(20 Points)** Using accumulator-style recursion, write a function `one-giant-string` that consumes a ListOfString and produces the concatenation of strings in the list in the order they appear in the list. That is, `(one-giant-string (list "Alice" "Bob" "Eve")` returns `"AliceBobEve"`.

What to Turn In

The rubric the graders will use for Homework 6 is posted below this assignment on Canvas. Programs must run in order to receive credit.

Note that code that is commented out will not be graded.

Using Canvas, turn in a single file containing all code and documentation for this assignment.

Name your file according to the naming conventions posts in the Assignments block on Canvas.

Make sure your name(s) and login(s)--for both partners, if applicable--appear at the top of the file in a comment.