CSE 341, Winter 2008, Assignment 1
Due: Thursday 17 January, 8:00AM

Last updated: January 14 (added hint to problem 6; changed solution length)

You will write 8 SML functions (plus 1 helper function) having to do with “lists of friends” like you might have on a social-networking site. The sample solution is roughly 75 lines.

See page 2 for additional instructions.

For this assignment, a “friend” is a pair of a string (the name) and an int (the age). An example would be ("Dan", 32). Though it shouldn’t make much difference, your functions must work for any string (even one with spaces, numbers, etc.) and any int (even a negative age). Most problems involves lists of friends, i.e., expressions of type (string*int) list.

1. Write a function hyaps that takes two friends and evaluates to true or false. It evaluates to true if the younger friend’s age is at least 7 greater than half the older friend’s age. To avoid rounding errors, do not use integer division or floating-point numbers. Do use integer multiplication, addition, and/or subtraction. (hyaps stands for “half your age plus seven.”) Note the younger friend might be either argument, and the two friends might have the same age.

2. Write a function all_older_than that takes a friend list and an age and evaluates to true or false. It evaluates to true if every friend in the list is strictly older than the age. Hint: If a list has no friends, then every friend in the list is older than any age.

3. Write a function make_identical_clique that takes a name, an age, and a clique size. It evaluates to a new friend list whose length is the clique size and whose members are all identical, with their name and age being those passed to make_identical_clique.

4. Write a function names_longer_than_age that takes a friend list and evaluates to another friend list that contains a subset of the friends in the input list. A friend should be in the output list if the size of the friend’s name (measured in characters) is strictly greater than the friend’s age. Use the SML library function String.size to produce a name’s size.

5. Write a function average that takes a friend list and evaluates to the average age of the friends in the list (rounded down as usual with integers). Your function should raise a “division by zero” exception if the friend list is empty. Hints: Write a helper function to compute the total age. Use the SML library function length. The syntax for integer division is e1 div e2.

6. Write a function oldest that takes a friend list and evaluates to a string option. It evaluates to NONE if the list has no friends and SOME n if the oldest friend in the list has name n. In case of ties, the name of the friend closest to the end of the list is the result. Hints: Write a helper function that returns a (string*int) option.

7. Write a function closest_to_age that takes a friend list and an age a and evaluates to another friend list. The result list should have all the friends whose ages are closest to a (so the result will have more than one friend only in the case of ties). Hints: Sample solution is 15 lines. Use the SML function abs to compute an absolute value. Having computed the result for a shorter list, you might use that result unchanged, add another friend to that result, or not include any friends from that result.

8. Write a function closest_to_average that takes a friend list and evaluates to another friend list containing all the friends whose ages are closest to the argument list’s average age. Hint: Use your earlier work!

9. (Challenge Problem) Your average function uses two recursive list functions (your helper function and length). Write average_challenge that produces the same result as average but uses only one call to one recursive list function. Hint: Have the helper function return a pair of integers.

10. (Challenge Problem) Write a function non_overlap that takes two friend lists and evaluates to a friend list containing exactly those friends that are in exactly one of the argument lists (not both). Hint: Use a couple helper functions.

Note: Remember the course policy on challenge problems.
Type Summary
Evaluating a correct homework solution should generate these bindings:

val hyaps = fn : (string * int) * (string * int) -> bool
val all_older_than = fn : (string * int) list * int -> bool
val names_longer_than_age = fn : (string * int) list -> (string * int) list
val make_identical_clique = fn : string * int * int -> (string * int) list
val names_longer_than_age = fn : (string * int) list -> (string * int) list
val average = fn : (string * int) list -> int
val oldest = fn : (string * int) list -> string option
val closest_to_age = fn : (string * int) list * int -> (string * int) list
val closest_to_average = fn : (string * int) list -> (string * int) list

Of course, generating these bindings does not guarantee that your solutions are correct. Test your functions.

Assessment
Your solutions should be:

• Correct
• In good style, including indentation and line breaks
• Written using the features we have used in class. In particular, you must not use references or arrays.
  (Why would you?)

Turn-in Instructions

• Put all your solutions in one file, lastname_hw1.sml, where lastname is replaced with your last name.
• The first line of your .sml file should be an ML comment with your name and the phrase homework 1.
• Go to https://catalysttools.washington.edu/collectit/dropbox/djg7/1359 (link available from the course website), follow the “Homework 1” link, and upload your file.
• If you have trouble accessing the web page for turning in your homework, contact Ben Lerner before the deadline. Emailing him your solution as an attachment is an undesirable back-up plan, and definitely should not be necessary after the first homework.

Syntax Hints
Small syntax errors can lead to strange error messages. Here are 3 examples for function definitions:

1. string * int list means string * (int list), not (string * int) list. You want (string * int) list for a friend list.
2. fun f x : t means the result type of f is t, whereas fun f (x:t) means the argument type of f is t. There is no need to write result types (and in later homeworks, no need to write argument types).
3. fun (x t), fun (t x), or fun (t : x) are all wrong, but the error message suggests you are trying to do something much more advanced than you actually are (which is trying to write fun (x : t)).