CSE 341, Winter 2008, Assignment 2
Due: Monday 28 January, 8:00AM

Last updated: January 20

You will write 11 SML functions (not counting local helper functions) relating to "contacts" and "announcements" like you might have on a social-networking site. You will also write an English description of SML code that computes, "reachable contacts." Your solutions must use pattern-matching. You may not use the functions `null`, `hd`, or `tl`, nor may you use anything containing a `#` character. You may not use mutation. The sample solution is about 180 lines, *including* all the code provided to you and not including comments. *Download hw2.sml from the course website.*

The provided code defines several types for you; you need not define any additional types.

- A `contact_category` describes how you might know someone: a "local friend," a "distant friend", a relative (which carries whether they are older or younger), or a coworker. A "contact" is a pair of a contact-category and a name (which is just a `string`).
- An `announcement` describes something you may want some contacts to know. There are "personal" messages, "professional" messages, and information about parties (with a message for what the party is about and a date for when it starts). There are also "forwarded" announcements that give the name of the contact who passed along some other announcement. Notice announcements can be forwarded multiple times.
- The `contact_count` type is used for the result in problem 1(e).

1. (Counting Contacts)
   (a) Write a function `num_informal1` that takes a list of contacts and returns how many contacts in the list can be "talked to informally." A contact can be talked to informally if and only if they are a friend (local or distant) or a younger relative. Do not use any helper functions.
   (b) Write a function `num_informal2` that behaves like `num_informal1` but uses a locally defined helper function that is tail recursive. Your helper function should take an accumulator argument.
   (c) Write a function `num_formal1` that takes a list of contacts and returns how many contacts in the list should be "talked to formally." A contact should be talked to formally if and only if they are an older relative or a coworker. You may use a helper function or not; your choice.
   (d) Write a function `names_informal` that takes a list of contacts and returns a list with the names of the contacts that can be "talked to informally." The order of the names does not matter.
   (e) Write a function `num_all_categories` that takes a list of contacts and returns a `contact_count`, where each field holds the number of contacts in the list in the corresponding category (with `friends` being the sum of local friends and distant friends). Hint: In the recursive case, use pattern matching to retrieve the values in a record produced from recursion.
   (f) Write a function `num_informal3` that behaves like `num_informal1` but uses `num_all_categories` instead of being recursive itself.
   (g) Write a function `num_formal2` that behaves like `num_formal1` but uses `num_all_categories` instead of being recursive itself.

2. (News Feeds)

   (a) Write a function `access_control` that takes a contact-category and an announcement and returns true if and only if that category "has the privilege of seeing" the announcement. Privilege rules are as follows:

   - Coworkers may not see personal announcements; everyone else can.
   - Only coworkers and older relatives may see professional announcements.
   - Only local friends may see party announcments.
   - Forwarded announcements have the same privilege as the announcement being forwarded.

   Use pattern-matching on the *pair* of arguments to produce a concise function.

   (b) Write a function `news_feed` that takes a contact-category and a list of announcements and produces a list holding the announcements in the input list that the contact-category has the privilege of seeing.

   (c) Write a function `all_news_feeds1` that takes a list of contacts and a list of announcements and returns a list of pairs. Each pair is the name of a contact in the contact list and the list of messages they have the privilege of seeing. Make `all_news_feeds1` a short recursive function using `news_feed` but no other helper functions.

   (d) Write a function `all_news_feeds2` that behaves like `all_news_feeds1` except it calls `news_feed` 5 times no matter how many contacts it is called with. Hint: Use 5 local bindings, *plus* a local helper function that processes the contacts list and uses the 5 local bindings.

3. (Reachable Contacts) *All you do for this problem is write comments explaining some provided code.* The provided function `reachable_contacts` takes a name (`name`) and a list of names and their "contacts" (`all_people`) and returns the list of names for which `name` is "connected" via some sequence of contacts. (In this problem and the challenge problem, "contacts" are just a list of names, with no mention of contact-categories.) Add comments *before* `reachable_contacts_helper` and `reachable_contacts` to explain *how* this code computes its result. Include in your description what each argument to `reachable_contacts_helper` contains during the computation and why `reachable_contacts` passes the arguments that it does. Explain why `reachable_contacts_helper` always terminates.

4. **(Challenge Problem)** Write a function `sorted_contacts` of type
   `name * (name * name list) list -> name list list`
   where `sorted_contacts(name,all_people)` produces a list where:

   - The first element is `[name]`
   - The second element is all of `name`'s contacts, but not including `name` (even if she has herself as a contact)
   - The third element is all of `name`'s contacts' contacts, but not including `name` or any of `name`'s contacts.
   - etc.

   In other words, the i^{th} element of the output is `name`'s "level-i" contacts, where the level of a contact is the length of the shortest path to them via contacts. The output should have length j if there is at least one "level-j" contact but no "level-j+1" contacts. Your output might have empty lists, but the last element should not be empty.

   Warning: The sample solution does not include the challenge problem.

Type Summary
Evaluating a correct homework solution should generate these bindings, in addition to the bindings from the code provided to you (but see the important caveat that follows!):

```
val num_informal1 = fn : contact list -> int
val num_informal2 = fn : contact list -> int
val num_formal1 = fn : contact list -> int
val names_informal = fn : contact list -> name list
val num_all_categories = fn : contact list -> contact_count
val num_informal3 = fn : contact list -> int
val num_formal2 = fn : contact list -> int
val access_control = fn : contact_category * announcement -> bool
val news_feed = fn : contact_category * announcement list -> announcement list
val all_news_feeds1 = fn : contact list * announcement list -> (name * announcement list) list
val all_news_feeds2 = fn : contact list * announcement list -> (name * announcement list) list
```

Important Caveat: The read-eval-print loop may give your functions *equivalent types* or *more general* types. This is fine. For example, for `contact list -> int`, equivalent types are `(contact_category * name) list -> int` or `(contact_category * string) list -> int` since type synonyms are equal to their definition. Also, `(contact_category * 'a) list -> int` is a more general type because a caller could still pass a `name` since `'a` can be any type. The sample solution, which omits most argument types, generates `(contact_category * 'a) list -> int` for several of the functions.

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), and using features we have used in class.

Turn-in Instructions

- Put all your solutions in one file, `lastname_hw2.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 2`.
- Go to `https://catalysttools.washington.edu/collectit/dropbox/djg7/1359` (link available from the course website), follow the "Homework 2" link, and upload your file.
- If you have trouble accessing the web page for turning in your homework, contact Ben Lerner *before* the deadline.