CSE 341, Spring 2008, Assignment 1
Due: Thursday 10 April, 8:00AM

Last updated: April 2 (correction to question 8)

You will write 11 SML functions having to do with calendar dates. In all problems, a “date” is an SML value of type int*int*int, where the first part is the year, the second part is the month, and the third part is the day. A “reasonable” date would have a positive year, a month between 1 and 12, and a day no greater than 31 (or less depending on the month). However, most problems do not assume “reasonable” dates; solutions should work for any int*int*int except where noted. A “day of year” is a number from 1 to 365 where, for example, 33 represents February 2. (We ignore leap years except in one challenge problem.)

The sample solution is roughly 75–80 lines. See page 2 for additional instructions.

1. Write a function is_older that takes two dates and evaluates to true or false. It evaluates to true if the first argument is a date that comes before the second argument. (If the two dates are the same, the result is false.)

2. Write a function number_in_month that takes a list of dates and a month (i.e., an int) and returns how many dates in the list are in the month.

3. Write a function number_in_months that takes a list of dates and a list of months (i.e., an int list) and returns the number of dates in the list of dates that are in any of the months in the list of months. Assume the list of months has no number repeated (or if a number is repeated then dates in that month are counted multiple times). Use your answer to the previous problem.

4. Write a function dates_in_month that takes a list of dates and a month (i.e., an int) and returns a list holding the dates from the argument list of dates that are in the month.

5. Write a function dates_in_months that takes a list of dates and a list of months (i.e., an int list) and returns a list holding the dates from the argument list of dates that are in any of the months in the list of months. Assume the list of months has no number repeated (or if a number is repeated then dates in that month are in the result list multiple times). Use your answer to the previous problem and ML’s list-append operator (@).

6. Write a function get_nth that takes a list of strings and an int n and returns the n'th element of the list where the head of the list is 1'st. If the list has too few elements, your function should apply hd to the empty list, which will raise an exception.

7. Write a function date_to_string that takes a date and returns a string of the form February 2, 2008 (for example). Use the operator ^ for concatenating strings and the library function Int.toString for converting an int to a string. For producing the month part, do not use a bunch of conditionals. Instead, use a list holding 12 strings and your answer to the previous problem.

8. Write a function number_before_reaching_sum that takes an int (which you can assume is non-negative) and an int list and returns an int. It returns n if sum is greater than or equal to the sum of the first n elements of the list, but not greater than or equal to the sum of the first n + 1 elements. If sum is greater than the sum of all numbers in the list, your function should apply hd to the empty list, which will raise an exception.

9. Write a function what_month that takes a day of year (i.e., a number between 1 and 365) and returns what month that day is in (1 for January, 2 for February, etc.). Use a list holding 12 integers and your answer to the previous problem.

10. Write a function month_range that takes two days of the year day1 and day2 and returns an int list [m1,m2,...,mn] where m1 is the month of day1, m2 is the month of day1+1, ..., and mn is the month of day day2. Note the result will have length day2 - day1 + 1 or length 0 if day1>day2.

11. Write a function oldest that takes a list of dates and evaluates to an (int*int*int) option. It evaluates to NONE if the list has no dates and SOME d if the date d is in the list and is older than all other dates in the list.
12. **Challenge Problem:** Write functions `number_in_months_challenge` and `dates_in_months_challenge` that are like your solutions to problems 3 and 5 except having a month in the second argument multiple times has no more effect than having it once. (Hint: Remove duplicates, then use previous work.)

13. **Challenge Problem:** Write a function `reasonable_date` that takes a date and determines if it actually describes a real date in the common era. A “real date” has a positive year (year 0 did not exist), a month between 1 and 12, and a day appropriate for the month. Your solution must handle leap years properly. Leap years are years that are either divisible by 400 or divisible by 4 but not divisible by 100.

Note: Remember the course policy on challenge problems.

**Type Summary**
Evaluating a correct homework solution should generate these bindings:

```ml
val is_older = fn : (int * int * int) * (int * int * int) -> bool
val number_in_month = fn : (int * int * int) list * int -> int
val number_in_months = fn : (int * int * int) list * int list -> int
val dates_in_month = fn : (int * int * int) list * int -> (int * int * int) list
val dates_in_months = fn : (int * int * int) list * int list -> (int * int * int) list
val get_nth = fn : string list * int -> string
val date_to_string = fn : int * int * int -> string
val number_before_reaching_sum = fn : int * int list -> int
val what_month = fn : int -> int
val month_range = fn : int * int -> int list
val oldest = fn : (int * int * int) list -> (int * int * int) option
```

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/djg7/2125](https://catalysttools.washington.edu/collectit/dropbox/djg7/2125) (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 Matthew Kehrt 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. `int * int * int list` means `int * int * (int list)`, *not* `int * int * int 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)`).