Assignment 0: Standard ML

CMPU-235: Programming Languages

Vassar College, **Fall 2017**

October 6, 11:59 p.m.

Description

You will write 12 SML functions (and tests for them) related to calendar dates. In all problems, a "date" is an SML value of type `int*int*int`, where the first part is the day, the second part is the month, and the third part is the year. A "reasonable" date has a positive year, a month between 1 and 12, and a day no greater than 31 (or less depending on the month). Your solutions need to work correctly only for reasonable dates, but do not check for reasonable dates (that is outside the scope of this assignment) and many of your functions will naturally work correctly for some/all non-reasonable dates. A "day of year" is a number from 1 to 365 where, for example, 33 represents February 2. *For the purpose of this assignment, we will ignore leap years.* Based on my sample solutions, your solution should be *approximately* 90–100 lines of code.

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 given 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.* Hint: *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. The returned list should contain dates in the order they were originally given.
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.* Hint: *Use your answer to the previous problem and SML'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}. Do not worry about the case where the list has too few elements: your function may apply `hd` or `tl` to the empty list in this case, which is okay.
7. Write a function `date_to_string` that takes a date and returns a `string` of the form `September-10-2015` (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. For consistency, use hyphens exactly as in the example and use English month names: January, February, March, April, May, June, July, August, September, October, November, December.
8. Write a function `number_before_reaching_sum` that takes an `int` called `sum`, *which you can assume is positive*, and an `int list`, *which you can assume contains all positive numbers*, and returns an `int`. You should return an int `n` such that the first `n` elements of the list add to less than sum, but the first `n` + 1 elements of the list add to `sum` or more. *Assume the entire list sums to more than the passed in value; it is okay for an exception to occur if this is not the case.*

9. Write a function `what_month` that takes a day of year (i.e., an `int` 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 else `SOME d` where the date `d` is the oldest date in the list.

12. Write a function `cumulative_sum` that takes a list of numbers and returns a list of the partial sums of these numbers. For example, `cumulative_sum [12,27,13] = [12,39,52]`. Hint: *Use a helper function that takes two arguments.*

Summary

Evaluating a correct homework solution should generate these bindings:

```
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
val cumulative_sum = fn : int list -> int list
```

Of course, generating these bindings does not guarantee that your solutions are correct. *Test your functions: Put your testing code in a separate file.* ***We will not grade the testing file, but you must turn it in.***

Turn-in Instructions

- Put all your solutions in one file, `assignment-00.sml`. Put tests you wrote in `assignment-00-tests.sml`. Put these two files in side a directory called `cmpu-235/assignments/00/`. *The reason for addng the course name is because some of you are in both of my classes. This will help prevent name collision.*
- Push the resulting code to the course GitHub repository: `http://github.com/vassar-cs/<github-username>`. *In case you don't yet have a repository, please contact me. Ideally, you should have accepted an invite that I sent a few weeks back.*

Assessment

The solutions for all the functions required in this assignment will be graded using the following criteria:

- **Correct (70 points)**
  The functions should pass all the tests that you provided with your solutions, and some test cases that I will run on the data.
- **Good Style (10 points)**
  The code should including proper indentation, reasonable variable names, line breaks and use *idiomatic* SML.
- **Documentation (10 points)**
  Should have appropriate documentation comments, and explicitly state any other assumptions that you might make in the process. **(10 points)**
- **Use a proper subset of SML (10 points)**
  Written using features discussed in class. In particular, you must *not* use SML's mutable references or arrays (Why would you?). It goes without saying, but you cannot use any existing library functions that handle dates. That will defeat the purpose of this assignment.

I realize that the second and third criteria are somewhat subjective, and will only remove points if those criteria are badly violated. For example, if you don't have any comments in the code, or if you write code which has a feel of the imperative style of coding. In other words, the points for the last three bullets are more or less yours to keep—if the solution is correct *and* if you have not made several deviations from the functional style of programming *and* if you use the allowable subset of SML.

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 assignments, 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)`).