CS 181, Winter 2022, Assignment 1
Due: Friday Jan 14, 11:00PM

You will write 11 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. You do *not* need to check for reasonable dates (that is a challenge problem). And, many of your functions will naturally work correctly for some/all non-reasonable dates. In the problems below, 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.)

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.) As an example, if the first parameter for a call to `is_older` is `(18,5,2019)`, and the second parameter is `(19,5,2019)`, the result should be `true`.

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} element. 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). You can 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 problem 6. Be sure to use hyphens exactly as in the given 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. You can 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.). Hint: use a list holding 12 integers and your answer to the previous problem. Remember we are ignoring leap years here.

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 so on, with the final element `mn` being the month of day `day2`. 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. (Remember you wrote an `is_older` function for the first problem.)

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 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. Solutions should properly handle leap years. Leap years are years that are either divisible by 400 or divisible by 4 but not divisible by 100. (Do not worry about days possibly lost in the conversion to the Gregorian calendar in the Late 1500s.)

Note: Remember the course policy on challenge problems.

Note: The sample solution contains *roughly* 90–100 lines of code, not including challenge problems.

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
```

Definitely check that the types of your bindings match the above. If the type of a binding doesn't match, your solution is guaranteed to be wrong. Of course, generating these bindings does not guarantee that your solutions are correct. You are free to have extra bindings in your file. *Test your functions: Put your testing code in a separate file. We will not grade the testing file, but you must turn it in.*

Assessment

Solutions should be:

- Correct
- In good style, including indentation and line breaks
- Written using features discussed in class. In particular, you must *not* use SML's mutable references or arrays. (Why would you?) Also do *not* use pattern-matching; it is the focus of the next assignment.

Turn-in Instructions

- Put all your solutions in a file named `hw1.sml`. Put the tests you wrote in a file named `hw1_test.sml`.
- Submit both files as your solution on Gradescope for Homework 1.

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)`).