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 \( \text{int}*\text{int}*\text{int} \), where the first part is the year, the second part is the month, and the third part is the day. 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 a challenge problem) 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. (We ignore leap years except in one challenge problem.)

1. Write a function \texttt{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 \texttt{number_in_month} that takes a list of dates and a month (i.e., an \text{int}) and returns how many dates in the list are in the given month.

3. Write a function \texttt{number_in_months} that takes a list of dates and a list of months (i.e., an \text{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. \textit{Assume the list of months has no number repeated}. Hint: Use your answer to the previous problem.

4. Write a function \texttt{dates_in_month} that takes a list of dates and a month (i.e., an \text{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 \texttt{dates_in_months} that takes a list of dates and a list of months (i.e., an \text{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. \textit{Assume the list of months has no number repeated}. Hint: Use your answer to the previous problem and SML’s list-append operator (\texttt{@}).

6. Write a function \texttt{get_nth} that takes a list of strings and an \text{int} \( n \) and returns the \( n \)th element of the list where the head of the list is 1\textsuperscript{st}. Do not worry about the case where the list has too few elements: your function may apply \texttt{hd} or \texttt{tl} to the empty list in this case, which is okay.

7. Write a function \texttt{date_to_string} that takes a date and returns a string of the form January 20, 2013 (for example). Use the operator \texttt{^} for concatenating strings and the library function \texttt{Int.toString} for converting an \text{int} to a string. For producing the month part, do \textit{not} use a bunch of conditionals. Instead, use a list holding 12 strings and your answer to the previous problem. For consistency, put a comma following the day and use capitalized English month names: January, February, March, April, May, June, July, August, September, October, November, December.

8. Write a function \texttt{number_before_reaching_sum} that takes an \text{int} called \texttt{sum}, which you can assume is positive, and an \text{int list}, which you can assume contains all positive numbers, and returns an \text{int}. You should return an int \( n \) such that the first \( n \) elements of the list add to less than \texttt{sum}, but the first \( n+1 \) elements of the list add to \texttt{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 \texttt{what_month} that takes a day of year (i.e., an \text{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 \texttt{month_range} that takes two days of the year \texttt{day1} and \texttt{day2} and returns an \text{int list} \([m1,m2,\ldots,mn]\) where \( m1 \) is the month of \texttt{day1}, \( m2 \) is the month of \texttt{day1+1}, \ldots, and \( mn \) is the month of day \texttt{day2}. Note the result will have length \texttt{day2 \texttt{ - } day1 + 1} or length 0 if \texttt{day1>day2}.

11. Write a function \texttt{oldest} that takes a list of dates and evaluates to an \text{(int*int*int)} option. It evaluates to \textit{NONE} if the list has no dates and \textit{SOME} \texttt{d} if the date \texttt{d} is the oldest date 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 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: **Challenge problems** are designed to have little (but some) impact on your grade whether you do them or not. They will typically allow you to approximately 5% of additional credit on an assignment, yet they will typically be much more difficult than this credit amount suggests. You should not attempt a challenge problem until you have finished the rest of an assignment.

Note: The sample solution contains roughly 75–80 lines of code, not including challenge problems.

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

```sml
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: 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 one file, `$yourIdNum$_.hw1.sml` (Sample, 5110000000_.hw1.sml). Put tests you wrote in `$yourIdNum$_.hw1_test.sml`.
- Upload your file to corresponding folders on `ftp://proglan:proglan@basics.sjtu.edu.cn`.
- If you have any trouble, contact the course staff before the deadline.

**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)`).