Instructions. Your task is to answer three written problems, and to write eleven SML functions related to calendar dates, as well as test cases for those functions. See the end of this document for instructions on how to submit your solution. In case you’re curious, our reference solution to the programming problems contains about 80 lines of code, excluding comments and blank lines.

Prohibitions. You may use only those SML features discussed so far in class. In particular, you may not use mutable references, arrays, pattern matching, or standard library functions not specifically mentioned in this homework.

On style. There is an ML style guide on the course website: [http://faculty.cs.gwu.edu/~clarkson/courses/csci4223/2013sp/sml-styleguide.pdf](http://faculty.cs.gwu.edu/~clarkson/courses/csci4223/2013sp/sml-styleguide.pdf) You need to consult it to complete the problems on this homework. Be aware that the style guide demonstrates some language features that you haven’t seen yet. Also, beware that the style guide recommends pattern matching (which you aren’t allowed to use in this homework) and recommends against `valOf`, `hd`, and `tl` (which you are required to use in this homework).

Part I: Written Problems

Hint: you might need to look up a things in an ML reference manual for this part—specifically, `char`, `real`, `length`, `str`, `chr`, and `floor`.

Problem 1. Give the types of each of the following ML values. For example, the type of `(1,2)` is `int * int`.

A. `["csci", 4223], ["csci", 6223]]`

B. `[[1,2],[],[3]]`

C. `(SOME 42, true)`

Give values that have the following ML types. None of your answers may include the empty list `[]`.

D. `(int * char) list`

E. `((int * real) list * string) list`

F. `int list list list list`

Problem 2. The following function executes correctly, but it was written with poor style. Your task is to rewrite it with better style. Please consult the style guide on the course website.

```sml
fun zardoz (a:int list, b:int list) =  
  if length(a) = 0 andalso length(b) = 0 then b  
  else if (length(b) = 0) then hd(a)::tl(a)  
  else if (length(a) = 0) then [] @ b  
  else if (hd(a) < hd(b)) = true then [hd(a)]  
  else if (hd(a) < hd(b)) = true then [hd(a)]  
  @ (zardoz((tl(a)), b)) else [hd(b)] @ (zardoz(a, (tl(b)))))
```
Problem 3. Complete the following comments by filling in the ellipses. At each place you fill in, give the complete environment.

```sml
val a = 3
(* Current static environment is a:int *)
(* Current dynamic environment is a-->3 *)
val b = 100.5
(* Current static environment is ... *)
(* Current dynamic environment is ... *)
val a = "three"
(* Current static environment is ... *)
(* Current dynamic environment is ... *)
val c = a^str(chr(floor(b)))
(* Current static environment is ... *)
(* Current dynamic environment is ... *)
```

Part II: Programming Problems

The TARDIS has been damaged, and the Doctor needs your help to reprogram its time circuitry. Many of the functions related to handling of dates need to be replaced. Allons-y!

Important Notes about Grading.

- Programs that do not compile will receive an automatic zero with no chance for a regrade. It’s better to comment out parts that don’t compile and submit a partial solution, rather than handing in a file that doesn’t compile.
- Function names and types are crucial, because we’re using automatic grading scripts. You must name functions and order the arguments as described below. You will receive automatic zeros for functions with improper names or arguments. Again, there will be no chance for a regrade—even if your function is otherwise properly written.
- Style matters. Unstylish code will lose points. Read the style guide at the course website.
- Review the policies on late submissions in the course syllabus. Be aware that if you submit version 1 of your solution on time, then submit version 2 after the deadline (even 1 minute after the deadline), we will grade version 2 and impose the appropriate late penalty. We will not grade version 1.

About Dates. For the purpose of this homework, a date-like triple is an SML value of type `int*int*int`. Examples of date-like triples include `(2013, 2, 1)` and `(0, 0, 1000)`. A date is a date-like triple whose first part is a positive year (i.e., a year in the common era), second part is a month between 1 and 12, and third part is a day between 1 and 31 (or 30, 29, or 28, depending on the month and year). `(2013, 2, 1)` is a date; `(0, 0, 1000)` is not. **Your functions need to work correctly only for dates**, not for arbitrary date-like triples. We will not test your functions using date-like triples that are not dates. Furthermore, your functions do not need to work correctly for leap years, except in one karma problem.

However, you will probably find it easier to write your solutions if you think about making them work for arbitrary date-like triples. For example, in your solution to the first problem below, it’s easier to forget about whether the input is truly a date, and simply write a function that claims (for example) that January 100, 2013 comes before February 34, 2013—because any date in January comes before any date in February.

A day of year is a number from 1 to 365 that represents the elapsed number of days in a year. For example, 33 represents February 2. (Again, we ignore leap years except in one karma problem.)
Functions to Write.

1. Write a function `is_before` 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 `count_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 that month. Note that the year is irrelevant to whether a date is in a month.

3. Write a function `count_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 contains no duplicates.** Hint: use your answer to the previous problem.

4. Write a function `dates_in_month` that takes a list `l` of dates, and a month, and returns the list of dates from `l` that are in the month. The dates should be in the same order as they originally appeared in `l`. (Note: your solution doesn’t need to name the list `l`; we’re just using that name to make the problem description clear. As far as our grading scripts are concerned, the types of arguments are important, but the names of arguments are not.)

5. Write a function `dates_in_months` that takes a list `l` of dates, and a list `ms` of months, and returns the list of dates from `l` that are in any of the months in the list of months. **Assume the list of months contains no duplicates.** The dates corresponding to the first month in `m` should appear first in your result list; the dates corresponding to the second month in `m` should appear next; etc. For each part of the list corresponding to a month `m`, the dates should be in the same order as they originally appeared in `l`. Use your answer to the previous problem and ML’s list-append operator `@`.

6. Write a function `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 defined to be the 1st element. If the list has too few elements, your function may apply `hd` or `tl` to the empty list, which will raise an exception.

7. Write a function `date_to_string` that takes a date, and returns a `string` representing that date in `middle endian` format with the month name spelled out. For example, the date `(2011,4,22)` would be represented as "April 22, 2011". There must be a comma after the day, a single space character in the two places shown in the previous examples, and the month name must be capitalized and properly spelled. Use ML’s built-in operator `^` (carat) for concatenating strings and standard library function `Int.toString` for converting an `int` to a `string`. For producing the month part, do not use conditional expressions. Instead, use a list holding twelve strings and your answer to the previous problem.

8. Write a function `elements_before_reaching_sum` that takes an `int` `s`, and an `int` list `l`, and returns an `int` `n`, such that the first `n` elements of `l` sum to strictly less than `s`, and that the first `n + 1` elements of `l` sum to `s` or greater. **Assume that `s` is positive, all the elements of `l` are positive, and that the entire list sums to greater than or equal to `s`.** If any of those assumptions fail to hold, it’s okay for your function to throw an exception. Hint: this is likely to be the algorithmically trickiest problem on this homework; test your solution well.

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 containing twelve integers and your answer to the previous problem.

10. Write a function `month_range` that takes two days of the year `d_1` and `d_2` and returns an `int` list that is `[m_1,m_2,...,m_n]`, where `m_1` is the month of `d_1`, `m_2` is the month of `(d_1 + 1)`, ..., and `m_n` is the month of day `d_2`. The length of the resulting list—that is, the `n` in `mn`—should be `d_2 - d_1 + 1`, or length 0 if `d_1 > d_2`.

11. Write a function `earliest` that takes a list `l` of dates, and evaluates to an `(int*int*int)` option. It evaluates to `NONE` if `l` is empty, and to `SOME d` if date `d` is the earliest date in the list.
12. **Karma Problem:** Write functions `count_in_months_karma` and `dates_in_months_karma` 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. **Karma Problem:** Write a function `is_date` that takes a date-like triple and determines if it describes a date, as defined at the beginning of this homework. If so, your function should return `true`; otherwise return `false`. Recall that a date has a positive year (year 0 CE does not exist), a month between 1 and 12, and a day appropriate for the month. Your solution must properly handle leap years. Leap years are years that are divisible by 400, or are divisible by 4 but not divisible by 100.

Note: Review the course policy (in the syllabus) on karma problems.

**Required bindings.** Evaluating a correct homework solution should generate (at least) the following bindings. To help you, we’ve supplied a template on the course website at [http://faculty.cs.gwu.edu/~clarkson/courses/csci4223/2013sp/hw1template.sml](http://faculty.cs.gwu.edu/~clarkson/courses/csci4223/2013sp/hw1template.sml).

```sml
val is_before = fn : (int * int * int) * (int * int * int) -> bool
val count_in_month = fn : (int * int * int) list * int -> int
val count_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 nth = fn : string list * int -> string
val date_to_string = fn : int * int * int -> string
val what_month = fn : int -> int
val month_range = fn : int * int -> int list
val earliest = fn : (int * int * int) list -> (int * int * int) option

Of course, generating those bindings does not guarantee that your solutions are correct...

**Testing.** You are required to test your functions. Put your testing code in a separate file. We will not directly grade it, but you must turn it in. Good test cases might help you get some partial credit if your solution is erroneous.

**Syntax Hints.** Small syntax mistakes can lead to strange error messages. Here are three common gotchas for function definitions:

- **int * int * int list** means `int * int * (int list)`, not `(int * int * int) list`.
- **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 in this homework. (And in later homeworks, there won’t even be a need to write argument types).
- **fun f (x t), fun f (t x), or fun f (t : x)** are all wrong, but the error message suggests you are trying to do something much more advanced than you probably are—which is trying to write `fun f (x : t)`.

**Submission Instructions**

Submissions that do not adhere to these criteria will lose points:

- Put all your written solutions to part I in one file, `netid_hw1written.txt`, where `netid` is your GW NetId (i.e., the part of your email address that comes before “@gwmail.gwu.edu”). Your GW NetId is **not** your GWID, which is a ‘G’ followed by eight digits. This file must be plain text. We recommend using Emacs to create it.
• Put all your solution code to part II in one file, `netid_hw1.sml`.
• Put all the tests you wrote for part II in another file, `netid_hw1_test.sml`.
• The first line of all three files should be an ML comment with your name, GW NetId, and the phrase `Homework 1`.
• Upload all three files to the Homework 1 assignment on BlackBoard.
• If you have trouble using BlackBoard, contact the course staff before the deadline.

Evaluation Criteria

Solutions will be evaluated on correctness with respect to the specifications in this assignment; style, including indentation and line breaks, with respect to the style guide on the course website; elegance, which is an ineffable quality that includes beauty, effectiveness, and simplicity; and adherence to using only those SML features discussed so far in class. In particular, you are prohibited from using mutable references, arrays, pattern matching, or standard library functions not specifically mentioned here. If you have questions about what is permitted and what is not, ASK.