CSC 330 —Programming Languages
Spring 2015
Assignment No. 1

Note 1 **This assignment is to be done individually**

Note 2 You can discuss the assignment with others, but copying code is prohibited.

- Due date: Jan 16, 2015, at the beginning of the class.
- This assignment is worth 1% of your total course mark.
- Submit electronically via Connex a single zip file with your solutions (two files)
- At the beginning of the class submit a paper copy of your program.

Objectives

After completing this assignment, you will have experience:

- Basic ML programming.

Your task, should you choose to accept it

You will write 12 ML functions (and tests for them) related to calendar dates. In all problems, a "date" is a ML value of type:

```
type DATE = {year:int, month:int, day:int}
```

Download the file `hw1.zip`. It contains two files, `hw1.sml` and `hw1-test.sml`. The first one will contain your code. The second contains test-cases.

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 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 only consider leap years in one function).

Part 1: Your 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 after 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 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 1st . Raise the exception `InvalidParameter` if n is zero or larger than the length of the list.

7. Write a function `date_to_string` that takes a date and returns a string of the form *January 20, 2013* (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, 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 `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 a DATE option. It evaluates to `NONE` if the list has no dates and `SOME d` if the date d is the oldest date in the list.

12. 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.)

The correct solution to the assignment should generate the following bindings:

```
type DATE = {day:int, month:int, year:int}
exception InvalidParameter
val is_older = fn : DATE * DATE -> bool
val number_in_month = fn : DATE list * int -> int
```

```
val number_in_months = fn : DATE list * int list -> int
val dates_in_month = fn : DATE list * int -> DATE list
val dates_in_months = fn : DATE list * int list -> DATE list
val get_nth = fn : string list * int -> string
val date_to_string = fn : DATE -> 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 : DATE list -> DATE option
val reasonable_date = fn : DATE -> bool
```

Having these bindings does not guarantee that your solution will be correct.

Part 2: Your Tests

I have provided a file with test bindings `hw1-test.sml`. Make sure that everyone of the test bindings evaluates to true. Add at least one new test for each function. Follow the convention used to name them. For example, the tests for the first function `is_older` have the prefix `test1` (e.g. `test1`, `test1a`, `test1b`, etc).

Hints

1. Implement one function at a time.
2. Remember that the REPL will keep all the bindings you define, and keep adding new ones every time you load the current file. I recommend you restart the REPL every time you change the datatype of a binding or its name.

Evaluation

Solutions should be:

1. Correct. We might use more tests than the ones provided. It will be run using SML/NJ.
2. In good style, including indentation and line breaks
3. Written using features discussed in class. In particular:
   - You must not use ML's mutable references.
   - You must not use arrays.
   - Also do not use pattern-matching; it is the focus of the next assignment.