CSE 341, Spring 2022, Homework 2

You will write 12 OCaml functions (and tests for them) related to calendar dates.

In all problems, a "date" is an OCaml 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. For example, `(1, 4, 2021)` represents the 1st day of the fourth month (April) in the year 2022.

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 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.)

To access elements of an date, use the provided functions `fst3`, `snd3`, and `thd3` to access the first, second, and third components accordingly. We will learn later how these functions work.

When writing your solution, feel free to refer to the library functions listed at the end of the homework. If you encounter an error saying "Unbound module Option", you probably have an old version of OCaml (4.07 or older). See the course website about how to upgrade your installation or how to upgrade on `attu`.

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 OCaml's list-append operator (`@`).

6. Write a function `get_nth` that takes a list of strings and a positive `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 `List.hd` or `List.tl` to the empty list in this case, which is okay.

7. Write a function `string_of_date` 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 `string_of_int` 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.

Note:

- There may be problems that have a corresponding function in the standard library. In that case, don't copy the standard library implementation, which will give you no points, because it uses the wrong language features that are not covered in class.
- The sample solution contains *roughly* 90–100 lines of code.

Summary

Evaluating a correct homework solution should generate these bindings:

```
val is_older : (int * int * int) * (int * int * int) -> bool = <fun>
val number_in_month : (int * int * int) list * int -> int = <fun>
val number_in_months : (int * int * int) list * int list -> int = <fun>
val dates_in_month : (int * int * int) list * int -> (int * int * int) list = <fun>
val dates_in_months : (int * int * int) list * int list -> (int * int * int) list = <fun>
val get_nth : string list * int -> string = <fun>
val string_of_date : int * int * int -> string = <fun>
val number_before_reaching_sum : int * int list -> int = <fun>
val what_month : int -> int = <fun>
val month_range : int * int -> int list = <fun>
val oldest : (int * int * int) list -> (int * int * int) option = <fun>
val cumulative_sum : int list -> int list = <fun>
```

Assessment

Each problem is worth 20 points: 10 points for the code and 10 points for the tests. Please also fill out `FEEDBACK.md` when you are done, for an extra 25 points. As a result, there are 265 points available this week, which is more than the 220 advertised on the syllabus. Because of this course's grading scheme, this works only to your advantage. You should feel free to skip a few problems if you want. If you receive more than 220 points, it will mean you will be able to skip more problems later in the quarter.

Syntax Hints

Small syntax errors can lead to strange error messages. Here are some examples.

1. `int * int * int list` means `int * int * (int list)`, not `(int * int * int) list`.

2. `let f(x : t, y : t)` will cause a syntax error in OCaml. The correct format is `let f((x : t), (y : t))`.

Library Functions

Here is a "cheatsheet" of library functions and operations that you may find helpful when completing your solution, in addtion to using the provided `fst3`, `snd3`, and `thd3`.

- `List.hd (* get the head of the list *)`

- `List.tl (* get the tail of the list *)`
- `@ (* infix operator to append two lists *)`
- `^ (* infix operator to concatenate two strings *)`
- `string_of_int (* convert an integer into a string *)`
- `Option.get (* if the argument = Some value, return value, otherwise raise an exception *)`
- `Option.is_some (* return true if the argument = Some value *)`
- `Option.is_none (* return true if the argument = None *)`