CSC 330: Assignment 1

Deadline: **Thu, Jan 22** at **23.59**

Submit via: CSC 330 Submission Portal

Note: the portal not yet operational; you will receive an announcement once the portal is up and running.

Ground rules

- You must complete this assignment *entirely* on your own. In other words, you should come up with the solution *yourself*, write the code *yourself*, and test the code *yourself*. The university policies on academic dishonesty (a.k.a. cheating) will be taken very seriously. Check the syllabus for details on what is allowed and what is not.
  - You can have high-level discussions with your classmates about the course material.
  - You are also welcome to use Teams or come to office hours and ask questions. If in doubt, please ask—we are here to help!
  - Using LLMs to get the solutions is akin to shooting yourself in the foot: you *might* pass the assignment, but you will most likely fail the written exams.
- You can use a maximum of **three (3) late days per assignment**, and a maximum of **seven (7) days for the whole semester**. Any further accommodations must be announced and negotiated with the instructor at least seven (7) days before the deadline.
- You will need to gather at least half of the total assignment score to pass the course.
- If you think some things should be explained in detail, please include the explanations in the form of source code comments.
- We will try to grade your assignments within seven (7) days of the initial submission deadline.
- If you think there is a problem with your grade, you have one week to raise concerns after the grades go public. TAs will be holding office hours during those seven days to address any such problems. After that, your grade is set in stone.
- You should only submit a single ZIP file containing two OCaml files in `.ml` format to the CSC 330 Submission Portal. The contents of the ZIP must include `prob1.ml` and `prob2.ml` (lowercase).
  - Warning: different file names might result in zero points. Our scripts expect the correct file names.
  - Another warning: put the filenames at the root of the ZIP file; if you put them within a directory, our script might not be able to find it.
  - **Important**: please submit the final ZIP, as well as the submission ID obtahined from the portal, via Brightspace (to ensure that there is a proper backup in case the submission portal suffers a failure).
- Document your code and include unit tests!
- Watch your style! We will penalize submissions with inadequate or bad style.
- Bonus problems can be used to improve your overall score; however, be aware that bonus problems might be harder than the rest. Bonus points cannot be transferred to other assignments or the final project.
- Your programs should generate **no** errors or warnings. If a program generates a single compilation error or warning, it will receive a zero.
- Please indicate your V-number and name within a comment at the top of each submitted file.
  - **Do not** include other private information (full name, your address, telephone number, medical exam tests etc.).

Furthermore, for this assignment:

- You cannot use mutation. Use of `ref`, `:=`, `!`, arrays, hashmaps and so on is forbidden.
- Make sure that your functions are named exactly as described. If your function is named `foo_v1` instead of `foo`, our test script will fail to find it, and you will get zero as a result.
- You cannot use the `open` statement.
- You cannot use any library function except the ones explicitly allowed for a given problem.
- You must not stray away from the features discussed in class. Do not use objects, modules and so on.

Test examples, updates and error corrections will be available on the CSC 330 Submission Portal. Make sure to check it from time to time.

Problem 1: Dates [25 points]

A date can be represented as a triple `(year, month, day)` of type `int * int * int`. Assuming that cool stuff like `match` and patterns do not exist, write the following date manipulation functions:

1. [5 points] `is_older` that takes two dates and returns true if the first date comes before the second date.
2. [5 points] `days_in_month` that takes a year and a month and returns the number of days in that month. Pay special attention to leap years. For the purpose of this problem, a leap year is a year that is divisible by 4, but not divisible by 100 unless it is divisible by 400.
3. [5 points] `dates_in_month` that takes a year and month and returns a list of valid dates in that month.
4. [5 points] `num_of_days` that takes in a date and outputs the number of days from the beginning of that year until (and including) the given date.
5. [5 points] `nth_day` that takes in a year and a number n and returns the date that corresponds to the n-th day of that year.

Rules

- You cannot use the `match` statement or patterns (e.g., `let x, y = z` is a no-no).
- As for the library calls:
  1. You can only use `Option.is_some`, `Option.is_none`, and `Option.get`, as well as `List.hd` and `List.tl`.
  2. You can also use the following functions for manipulating triplets (copy/paste them to the beginning of your file):

```
let fst3 (x, _, _) = x
let snd3 (_, x, _) = x
let thd3 (_, _, x) = x
```

Hints

- A date is valid if:
  i. $1 \leq \text{year} \leq 3000$,
  ii. $1 \leq \text{month} \leq 12$, and
  iii. the day is a valid day in a given month. All subproblems except the first one should return `None` if a date is invalid, and `Some val` if a date is valid.
- The expected type signatures are:

```
val is_older : (int * int * int) * (int * int * int) -> bool
val days_in_month : (int * int) -> int option
val dates_in_month : (int * int) -> (int * int * int) list option
val num_of_days : (int * int * int) -> int option
val nth_day : (int * int) -> (int * int * int) option
```

Problem 2: Inflation [25 points]

We are living in times of high and increasing inflation rates. That should be quite obvious by now (if you don't believe me, take a stroll across a grocery store of your preference then). However, it might not be obvious whether this is an abberation or a return to the normal state of the world. Why don't we try to analyze some data and resolve this conundrum?

The World Bank— love them or hate them— has kept a record of inflation rates since 1960. Let's pretend to believe that they did a good job, and let's analyze that data— with OCaml, of course.

Input file format

The dataset in the CSV format is available here. The rows stand for different countries, and columns in each row are separated with a comma (`,`). Each row contains three or more columns. The columns are as follows:

1. Country name in plain English
2. Three-letter country identifier (e.g., `CAN` for Canada)
3. 1960 inflation rate
4. 1961 inflation rate
5. 1962 inflation rate …

You can assume that all records in the file are all valid (no row will have zero columns etc.). You can furthermore assume that a valid rate is a parsable floating point number. However, because the World Bank is not omnipotent, despite the rumours, the yearly inflation rates for many countries are missing. In such cases, the rate is just an empty string or another non-parsable float.

Tasks

1. [5 points] Write a function `get_records` that takes in the file contents as a string, and returns a list of rows. Each row is represented as a record consisting of the country identifier (named `id`), country name (named `name`), and a list of inflation rates (named `rates`). If the inflation rate is not available for a given year, you should use `None`.
   - Hint: Use `String.split_on_char` and `Float.of_string_opt` for this problem.
2. [5 points] Write a tail-recursive function `avail` that, for a given country record, returns the number of available yearly records.
3. [5 points] Write a tail-recursive function `last` that, for a given country record, returns the most recent year for which the inflation rate is available, together with its inflation rate.
4. [5 points] Write a tail-recursive function `minmax` that, for a given country record, returns the tuple consisting of:
   - a year and a rate for which the inflation was the lowest, and
   - a year and a rate for which it was the highest.
5. [5 points] Write a function `summarize` that takes the list of all countries and a single country identifier, then finds the country with the given identifier in the list and summarizes the found country as follows:

   ```
   Country: Neverland (NVL)
   Records available: 26 years
   Last record: 2021 with rate of 0.1%
   Lowest rate: 2018 with rate of -1.1%
   Highest rate: 2019 with rate of 105.1%
   ```

   The summary is returned as a single string. Return `Cannot find XXX` if a country cannot be found, where `XXX` is the provided ID. Make sure to use the functions from the previous steps for this task.

Bonus tasks

6. [5 points] Write a tail-recursive function `concat` that concatenates a list of strings into a single string separated by the first argument. Empty strings should be ignored.
   - Example: `concat (", ", ["a"; ""; "b"])` should return `"a, b"`
   - Note: pay special attention to corner cases!

Rules

- In contrast to Problem 1, here you cannot use the following functions: `List.hd`, `List.tl`, `Option.get`, `Option.is_some`, and `Option.is_none`.

    – Use pattern matching instead!
- You can, however, use `String.split_on_char`, `Float.of_string_opt`, `string_of_float` and `string_of_int`.
- Use of operator `^` is allowed.

Hints

- Use `[@tailcall]` to test if your function is tail-recursive.
- You can use the following function to read the content of a file into a string:

```
let read_file path =
  let fp = open_in path in
  let s = really_input_string fp (in_channel_length fp) in
  close_in fp;
  s
```

- The expected type signatures are:

```
type country = { id : string; name : string; rates : float option list }
val get_records : string -> country list
val avail : country -> int
val last : country -> (int * float) option
val minmax : country -> (int * float) option * (int * float) option
val summarize : country list * string -> string

(* Bonus *)
val concat : string * string list -> string
```

Testing

Suppose that you want to test the first problem (`prob1.ml`). You should create a file `test_prob1.ml` with the following contents:

```
open Prob1  (* this is the _only_ place you should use `open` in this assignment *)

let test_is_older () =
  assert (is_older ((1990, 1, 1), (1995, 1, 15)) = true);
  assert (is_older ((1996, 2, 21), (1996, 2, 20)) = false)
let _ = test_is_older ()

(* you should create similar functions with many test cases for other tasks as well *)
```

Once done, you can create an executable with the following command:

```
ocamlc -o prob1 prob1.ml test_prob1.ml  # This compiles source files into an executable
./prob1                                 # This line executes the compiled executable
```

If any of your tests fail, you will get the following error:

```
Fatal error: exception Assert_failure("test_prob1.ml", 5, 2)
```

indicating that you have a problem at line 5 of `test_prob1.ml`.

You can also use utop if you prefer to test stuff manually. Another idea is to use a proper unit testing framework such as OUnit (details are available here). Note that the latter choice is recommended only for adventurous souls— it goes way beyond the stuff we've learned thus far.

Sample test cases

```
is_older ((1990, 1, 1), (1992, 1, 1))
(* true *)

days_in_month (2023, 1)
(* Some 31 *)
days_in_month (2023, 15)
(* None *)

dates_in_month (2023, 1)
(* Some [(2023, 1, 1), (2023, 1, 2), ...] *)
dates_in_month (2023, 15)
(* None *)

num_of_days (2023, 2, 1)
(* Some 32 *)

nth_day (2023, 32)
(* Some (2023, 2, 1) *)


let data = "Canada,CAN,0.1,,2.1\nUnited States,USA,1.1,2.1,1.1\nUnknown,UNK,,,"
let records = get_records data
(* [ {id = "CAN"; name = "Canada";        rates = [Some 0.1; None; Some 2.1]}
   ; {id = "USA"; name = "United States"; rates = [Some 1.1; Some 2.1; Some 1.1]}
   ; {id = "UNK"; name = "Unknown";       rates = [None; None; None]} ] *)

avail (List.hd records)
(* 2 *)

last (List.hd records)
(* Some (1962, 2.1) *)
last {id = "UNK"; name = "Unknown"; rates = [None; None; None]}
(* None *)
```

```
minmax (List.hd records)
(* (Some (1960, 0.1), Some (1962, 2.1)) *)

summarize (records, "USA")
(*
Country: United States (USA)
Records available: 3 years
Last record: 1962 with rate of 1.1%
Lowest rate: 1962 with rate of 1.1%
Highest rate: 1961 with rate of 2.1%
*)
summarize (records, "UNK")
(*
Country: Unknown (UNK)
Records available: 0 years
*)
summarize (records, "EEE")
(* Cannot find EEE *)
```