Homework 3

Functional Programming (ITI0212)

due: 2022-04-15

Place your solutions in a module named `Homework3` in a file with path `homework/Homework3.idr` within a repository called `iti0212-2022` on the TalTech GitLab server (`https://gitlab.cs.ttu.ee/`). Your solutions will be pulled automatically for marking shortly after the due date.

At the start of the file include a comment containing your name as it appears in your university records. Precede each problem's solution with a comment specifying the problem number.

The solution file that you submit should load without errors. If you encounter a syntax or type error that you are unable to resolve, please use comments or holes to isolate them from the part of the file interpreted by Idris.

Problem 1
Write a named implementation of the `Ord` interface for `String`s that is a case-insensitive version of the implementation `Ord String` in the standard library, i.e. a case-insensitive lexicographical order. For example, `compare @{ci} "aBc" "abc"` should evaluate to `EQ`.

Problem 2
Write a record type `Book`, with fields for title, author, and publication year, and write a record type `Library`, with a field for the name of the library, and a list of books.

Problem 3
Implement the `Eq` and `Ord` interfaces for `Book`. One book should equal another only when all of the fields are equal. Books should be ordered by author first, then title, then year.

Problem 4
Write a function `add_book` that adds a `Book` to a `Library`, and a function `find_title` to search a `Library` by book title, returning the first matching `Book` (if there is one), according to the ordering from Problem 3.

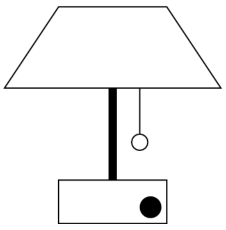
Problem 5
Write a function `every_other` that produces a new `Stream` from every second element of an arbitrary input `Stream`.

For example, `take 5 (every_other [1..])` should evaluate to `[2, 4, 6, 8, 10]`.

Problem 6
A *deterministic labelled transition system* can be represented using records and co-data.

It consists of a type of *states*, a type of *labels*, and a *transition function* that takes a current state, a potentially infinite sequence of labels, and returns a potentially infinite sequence of states (the sequence of states visited by the transition system in processing the input labels).

For example, consider the following labelled transition system representing a lamp. There are three states (`Off, On, Broken`), three values for labels (`Pull, Reset, Break`), and the arrows show what happens when the system processes a label in a state (for example, `Pull` in the state `Off` causes the system to move to the state `On`). The state should be unchanged if there is no appropriate label, for example if the system is `Broken` then it should remain `Broken` if we try to process `Break`.



$$\begin{tikzcd}
\text{Off} \arrow[rr, shift left, "pull"] & & \text{On} \arrow[ll, shift left, "pull"] \arrow[dl, bend left, "break"] \\
& \text{Broken} \arrow[ul, bend left, "reset"] \arrow[loop below, "pull"] &
\end{tikzcd}$$

Write a record `DLTS` to represent deterministic labelled transition systems, and write a term `lamp : DLTS` representing the system shown above.

For example, if your field for the transition function is called `transition` then

`take 3 (lamp.transition Broken (Reset :: (repeat Pull)))`

should evaluate to `[Off, On, Off]`, and

`take 1 (lamp.transition On [Reset])`

should evaluate to `[On]`.

Problem 7
Consider the following "fast" function for calculating the n^{th} Fibonacci number.

```
fibStream : Stream (Pair Integer Integer)
fibStream = (0 , 1) :: map next fibStream
  where
    next : Pair Integer Integer -> Pair Integer Integer
    next (fib_pp, fib_p) = (fib_p, fib_pp + fib_p)

nth_fib : Nat -> Integer
nth_fib n = (fst . head . drop n) fibStream
```

(We use `Integer` instead of `Nat` since integer arithmetic is much faster than natural number arithmetic, in Idris)

The idea behind this function is that if we have the $(n-2)^{\text{th}}$ and $(n-1)^{\text{th}}$ Fibonacci number then we can quickly compute the n^{th} Fibonacci number by just adding these. `fibStream` is a stream of pairs of consecutive Fibonacci numbers calculated in this way, and so the n^{th} Fibonacci number is just the first element of the pair at the n^{th} index of the stream.

Idris does not recognize `nth_fib` as total, because it does not recognize `fibStream` as total. However, we know that in reality it is total (it produces a non-empty finite prefix of the stream in a finite time), and furthermore we know that `nth_fib` is total, and would like this to be known to Idris and so to other users of our code.

Recalling Idris' syntactic conservative approximation to productivity, rewrite `fibStream` such that Idris recognizes it as total (and so consequently recognizes `nth_fib` as total).