Lab 6

Due Sunday, October 29 by 10:00pm

Handout 20
CSCI 334: Fall 2023

Turn-In Instructions

Part of this assignment must be written using LaTeX. I provide a LaTeX template in your repository for you to get started. The template I provide compiles without error as-is. For full credit, you must submit both your `.tex` source file as well as the rendered `.pdf` file. Your source file should be called `lab-6.tex` and your PDF should be called `lab-6.pdf`. (5 points)

For each programming question in this assignment, create a project directory. For example, the source directory for question 2 should be in a folder called "`q2`". You should be able to `cd` into this directory and then run the program by typing the command "`dotnet run`", with additional arguments depending on the question.

Each program should be split into two pieces: a "`Program.fs`" file that contains the `main` method and associated program-startup helpers (if needed), and another "`Library.fs`" file that contains the function(s) of interest in the question. All library code should be in a module named "`CS334`". For full credit, your program should both build and run correctly. Be sure to provide usage output (defined in `main`) for all programs that require arguments, and be sure to validate the input that the user gives you.

Turn in your work using the `git` repository assigned to you. The name of the repository will have the form `https://aslan.barowy.net/cs334-f23/cs334-lab06-<USERNAME>.git`. For example, if your username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-f23/cs334-lab06-abc1.git`.

Honor Code

This is a solo lab. You may work with another classmate to understand what the problems ask, but you are not permitted to develop solutions together. Submitted solutions must be exclusively your own. Please refer to the section "single author programming assignments" in the honor code handout for additional information. You do not need to submit a `collaborators.txt` file for this assignment. You are always welcome to ask me for clarification if the above is unclear in some circumstance.

This assignment is due on Sunday, October 29 by 10:00pm.

Sanity Check: Students sometimes submit incomplete assignments, accidentally forgetting to run `git add` for all of their files. Fortunately, there is an easy way to make sure that this does not happen to you. Before you are done, `git clone` your repository to a new folder and then try building/running everything. It only takes a couple minutes and can spare you from headaches later on.

Reading

1. **(Required)** "How to Fix a Motorcycle"
2. **(As needed)** Refer to previous readings on F#.

Problems

Q1. (50 points) ..... How to Fix a Motorcycle

Read the excerpt of Zen and the Art of Motorcycle Maintenance titled "How to Fix a Motorcycle" by Robert Pirsig, and then answer the following questions. Please try to keep your responses short. For all three questions below, you should write between 200 and 400 words total.

(a) Think back about your experiences doing programming in the past. Think of a time when it was difficult and you were feeling demoralized. This may have been a programming assignment, programming for work, or programming for fun. Now that you've read Pirsig's essay, do any of his "gumption traps" apply to your experience? Describe your experience.

(b) Whether Pirsig's advice applies to your situation or not, what would you do differently now if you were in that situation again?

(c) To what extent do you think your situation could have been improved by a better programming language? In other words, suppose your programming language helped you more. What pitfalls or gumption traps might be avoided? Don't worry too much about whether your imagined features are impractical to implement.

Q2. (25 points) ..... Partial Application

In this problem, we will use currying and partial application to produce convenience functions.

(a) Begin by writing a logging function. This function is intended to be called by another program and should write a message to the given file. Here is a sample output.

```
[2023-10-23 08:20][mysqld] FAIL: Tried (and failed) to read database too many times. Quitting.
```

Here is another sample output.

```
[2022-12-30 21:53][logind] WARN: User 'dbarowy' attempted to login without password.
```

Observe that the above string has four fields:

- The date, in `yyyy-MM-dd HH:mm` format.
- The name of the program (the "daemon") calling the log function.
- The severity of the message (which is either `INFO`, `WARN`, or `FAIL`).
- A message.

Your `log` function should have the following declaration.

```
let log (date: System.DateTime)(severity: string)(daemon: string)(message: string)(file: string) : unit = ...
```

To write to a file, `log` should use the `System.IO.File.AppendAllText` function (see `File.AppendAllText` documentation). You will also need to convert the `System.DateTime` object into a `string` having the format shown above (see `System.DateTime` documentation). Hint: if you find this to be difficult, be sure to look at `DateTime`'s `ToString` method.

(b) Next, write the following convenience function.

```
let lognow = ...
```

Importantly, `lognow` should be defined only by partial application of `log`. Specifically, `lognow` should be called `log` with a single argument corresponding to the current `DateTime` (see the documentation for `Now`). If you have done this correctly, then the return type for `lognow` will be `string -> string -> string -> string -> unit`.

(c) Next, write the following three convenience functions.

```
let info = ...

let warn = ...

let fail = ...
```

Each definition should be defined by calling `lognow` with a single argument corresponding to the `severity` parameter. If you have done this correctly, then the return type for each of the above functions will be `string -> string -> string -> unit`. You should be able to call the above functions like so,

```
info "daemon" "message" "foobar.txt"
```

and the effect will be that text resembling the following will be written to the file `foobar.txt`:

```
[2023-10-23 08:20][daemon] INFO: message
```

(d) Finally, make it so that the user can call your program from the command line with an expression of the form:

```
$ dotnet run <severity> <daemon> <message> <file>
```

The project directory for this question should be called "`q2`". You should be able to run your program on the command line by typing, for example, "`dotnet run warn sshd "brute force attack detected" sshd-log.txt`".

Q3. (20 points) ..... Cartesian Product

Write an F# function `cproduct` that computes the Cartesian product of two lists. The Cartesian product is defined as

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

Define your function however you like (e.g., using `rec` or `List.map`), but it must have the following type signature:

```
cproduct: 'a list -> 'b list -> ('a * 'b) list
```

Note that because we are using a `list` instead of a `set` data type, duplicate values will not be handled correctly. For this assignment, do not worry about duplicate values.

In your `main` method, compute the following:

```
let xs: (char * int) list = cproduct ['a'; 'b'; 'c'; 'd'] [1; 2; 3; 4]

let ys: (char * int) list = cproduct ['a'; 'b'; 'c'; 'd'] []

let zs =
    cproduct
        ['a'; 'b'; 'c'; 'd']
        ([1; 2; 3; 4] |>
            List.map
                (fun _ ->
                   System.Threading.Thread.Sleep(1000)
                   System.DateTime.Now
                )
        )
```

The project directory for this question should be called "`q3`". You should be able to run your program on the command line by typing "`dotnet run`".

Q4. ($\frac{1}{10}^{\text{th}}$ bonus point) ..... Optional: Feedback

I always appreciate hearing back about how easy or difficult an assignment is.

For $\frac{1}{10}^{\text{th}}$ of a bonus to your final grade, please fill out the following Google Form.