Lab 5

Handout 12

Due Monday, November 11 by 11:59pm

CSCI 334: Spring 2024

Coding Guidelines

Each question in this assignment should go into the appropriate project directory. For example, the solution to question 1 should be in a folder called "`q1`". When a solution is a program, one should be able to `cd` into the question directory and then run your program by typing the command "`dotnet run`", with additional arguments depending on the question.

Every 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. Library code should be contained within a module named "`CS334`". Be sure to provide usage output (defined in `main`) for all programs that require arguments. For full credit, your program should both build and run correctly.

If any of your programs take input from the user, be sure that your program validates input: when a user fails to supply input, or supplies input that does not make sense, your program should print a usage message and return with a nonzero exit code. Users should never experience a program crash in this class; exceptions should be prevented from arising or be caught whenever bad input is encountered. Think through problem corner cases carefully.

Turn-In Instructions

Turn in your work using the `git` repository assigned to you. The name of the `git` repository will have the form `https://aslan.barowy.net/cs334-s24/cs334-lab05-<USERNAME>.git`. For example, if your CS username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-s24/cs334-lab05-abc1.git`.

You should have received an invite to commit to the repository via email. If you did not receive an email, please contact me right away!

Group Programming Assignment

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. Tell me who your partner is by committing a `collaborators.txt` file to your repository. **Be sure to commit this file whether you worked with a partner or not.** If you worked by yourself, `collaborators.txt` should contain something like "I worked by myself." (5 points)

This assignment is due on Monday, November 11 by 11:59pm.

Reading

1. **(Required)** "Higher-Order Functions"
2. **(As needed)** Microsoft's Official F# Documentation

Problems

Q1. (35 points) ..... Mapping Functions

Write a function `censor` that takes a list of banned words and a list of words to potentially censor.

```
let censor (banned: string list)(words: string list) : string list = ...
```

Censored words are replaced with `XXXX`. For example,

```
> censor
-   ["party"; "hoxsey"]
-   ["we're"; "going";"to";"skip";"class";"for";"a";"party";"on";"hoxsey"]
- ;;
val it: string list =
  ["we're"; "going"; "to"; "skip"; "class"; "for"; "a"; "XXXX"; "on"; "XXXX"]
```

Your `censor` function must use `List.map`, making use of another function `memberOf`. `memberOf` is a recursive function that takes a list of banned words and a single word to potentially censor, returning `true` if the word is in the banned list and `false` otherwise. Censoring should work regardless of case (i.e., "hoxsey" and "HOXSEY" should be considered the same).

```
let rec memberOf(banned: string list)(word: string) : bool =
```

`memberOf` should not call any other functions except that you may use the following case-insensitive string comparison function,

```
System.String.Equals(s1, s2, System.StringComparison.CurrentCultureIgnoreCase)
```

where `s1` and `s2` are strings.

You should be able to run your program on the command line by supplying a path to a banned word list and a sequence of banned words.

```
$ dotnet run banned.txt I need an extension because I got lost in the steam tunnels
I need an XXXX because I got lost in the XXXX XXXX

$ dotnet run banned.txt I like programming languages more than the fried rice at Blue Mango
I like programming languages more than the XXXX XXXX at XXXX XXXX
```

If the program is run without any arguments, or if the banned file does not exist, it should print out the following usage string and quit with exit code 1:

```
Usage: <banned.txt> <word_1> [... <word_n>]
```

Here are some tips for making the above work.

To read in a file, you can use the following construct, which opens `filename` and returns a list of strings, one string for each line of the file.

```
IO.File.ReadLines(filename) |> Seq.toList
```

If `filename` does not exist, the above will throw `System.IO.FileNotFoundException` exception. Prevent that from happening by using the `System.IO.File.Exists` method or just handle the exception when it is raised.

To convert an array to a list, use the `Array.toList` function.

F# has array-slicing capabilities that let you easily take a subset of an array, just like in Python. See the F# documentation.

Finally, a list of strings `strs` can be concatenated into a single string with a separator of your choice (e.g., `" "`) like so:

```
System.String.Join(" ", strs)
```

Which words to include in your banned word list is up to you, however, be sure to include at least the set of words that make the above examples work.

The project directory for this question should be called "`q1`". Your `censor` and `memberOf` functions should be in a module called `CS334` stored in a file called `Library.fs` and your `main` function should be in a file called `Program.fs` as in previous labs. As usual, write your program to guarantee that user-provided input makes sense and does not throw an exception.

Q2. (20 points) ..... F# Map for Trees

(a) The binary tree datatype

```
type Tree<'a> =
| Leaf of 'a
| Node of Tree<'a> * Tree<'a>
```

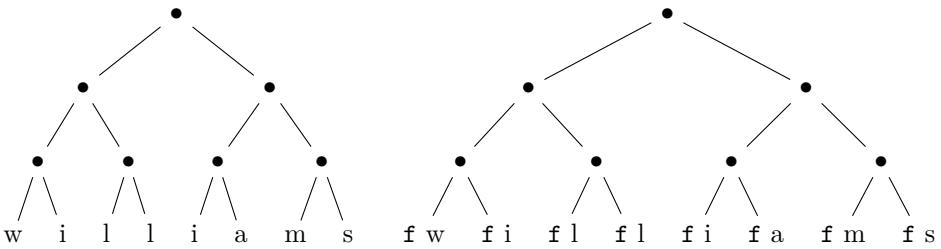
describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node).

Write the function

```
let rec maptree f t = ???
```

where `f` is a function and `t` is a tree. `maptree` should return a new tree that has the same structure as `t` but where the values stored in `t` have the function `f` applied to them.

Graphically, if `f` is a function that can be applied to values stored in the leaves of tree `t`, and `t` is the tree on the left, then `maptree f t` should produce the tree on the right.



For example, if `f` is the function `let f x = x + 1` then

`maptree f (Node(Node(Leaf 2, Leaf 3), Leaf 4));;`

should evaluate to `Node (Node (Leaf 3,Leaf 4),Leaf 5)`.

(b) In a comment block above your `maptree` definition, explain your definition in one or two sentences. Comment blocks in ML look like the following.

```
(*
 * Says hello to the given name.
 *
 * @param   name The name.
 * @return       Nothing.
 *)
let sayHello name =
    printfn "Hello %s!" name
```

Be sure to provide `@param` and `@return` tags.

(c) What type does F# give to your function? Why isn't it the type `('a → 'a) → Tree<'a> → Tree<'a>`? Provide an answer in the comment block of your `maptree` function.

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`" and output like the kind shown above should be printed to the screen. Be sure to provide several examples that demonstrate that your function works correctly.

Q3. (20 points) ..... F# Reduce for Trees

The binary tree datatype

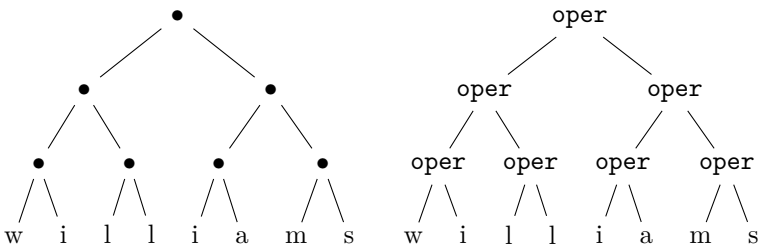
```
type Tree<'a> =
| Leaf of 'a
| Node of Tree<'a> * Tree<'a>
```

describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node).

(a) Write a function

`treduce : ('a → 'a → 'a ) → Tree<'a> → 'a`

that combines all the values of the leaves using the binary operation passed as the first parameter. In more detail, if `oper : 'a → 'a → 'a` and `t` is the nonempty tree on the left in this picture,



then `treduce oper t` should be the result obtained by evaluating the tree on the right. For example, if `f` is the function

```
let f x y = x + y
```

then `treduce f (Node(Node(Leaf 1, Leaf 2), Leaf 3))` $= (1 + 2) + 3$ and the output is `6`.

(b) In a comment block above your `treduce` definition, explain your definition of `treduce` in one or two sentences. Be sure to provide `@param` and `@return` tags.

The project directory for this question should be called "`q3`". You should be able to run your program on the command line by typing, for example, "`dotnet run`" and output like the kind shown above should be printed to the screen. Be sure to provide several examples that demonstrate that your function works correctly.

Q4. (20 points) ..... Cartesian Product

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

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

`cproduct` must have the following type signature:

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

Because we are using a `list` instead of a `set` data type, it is possible that your algorithm will generate duplicate values. Do not worry about duplicate values for this assignment.

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 "`q4`". You should be able to run your program on the command line by typing "`dotnet run`".

Q5. ($\frac{1}{10}^{\text{th}}$ 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.