Homework 5

Due Sunday, October 16 by 10:00pm

Handout 13
CSCI 334: Fall 2022

Turn-In Instructions

Written questions in this assignment must be written using LaTeX. I provide a LaTeX template in your repository for you to use to get started.

Please note that for full credit, you must submit both your `.tex` source file as well as the rendered `.pdf` file. (5 points) Your source file should be called `lab-5.tex` and your PDF should be called `lab-5.pdf`. (5 points)

For programming questions, create a project directory. For example, the source directory for question 1 should be in a folder called "`q1`". 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. As before, programs should be split into two pieces: a "`Program.fs`" file that contains the `main` method, and a "`Library.fs`" file that contains the functions of interest in the question. All library code should be in a module named "`CS334`".

Turn in your work using the Gitlab repository assigned to you. The name of the Github repository will have the form `https://evolene.cs.williams.edu/cs334-f22/<YOUR_USERNAME>/lab05.git`. For example, if your CS username is `22abc1`, the repository would be `https://evolene.cs.williams.edu/cs334-f22/22abc1/lab05.git`.

Honor Code

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. **Be sure to tell me who your partner is** by committing a `collaborators.txt` file to your repository (5 points).

This assignment is due on Sunday, October 16 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)** "Proof by Reduction"

Problems

Q1. (40 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) = ...
```

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) =
```

`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
```

Note that if `filename` does not exist, the above will throw `System.IO.FileNotFoundException` exception.

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. 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.

Q2. (20 points) ..... Halting on any input

The function $\texttt{Halt}_\forall$ (pronounced "halt-any") is defined as:

$$\texttt{Halt}_\forall(\texttt{p}) = \begin{cases} \texttt{true} & \text{if p halts on any input.} \\ \texttt{false} & \text{otherwise.} \end{cases}$$

Assume `p` is a `String` representation of a Python function always having the form:

```
def prog(x):
    # whatever
```

and that `x` is just an ordinary primitive value, like an integer.

Prove that $\texttt{Halt}_\forall$ is not computable.

Q3. (30 points) ..... Garbage Collection

A garbage collection algorithm performs automatic cleanup of unused memory in a program. Modern programming language runtimes routinely perform garbage collection in order to dramatically simplify memory management. Garbage has the following definition.

> At a given point i in the execution of a program P, a memory location m is garbage if continued execution of P from i **will not** access location m again.

Nonetheless, garbage collection using the above definition of garbage is not computable. Instead, languages solve a simpler problem by using a slightly different definition of garbage:

> At a given point i in the execution of a program P, a memory location m is definitely garbage if continued execution of P from i **cannot** access location m again.

McCarthy's "mark sweep" algorithm uses this latter definition, because it only reclaims memory that is impossible to re-read.

Prove that garbage collection using the first definition is not computable. You should prove this fact using the "reductio ad absurdum" proof technique. Specifically, your proof should employ a reduction of another non-computable function to garbage collection. For example, you may rely on the fact that we know that the halting problem is not computable.

Assume that you have the following `isGarbage` function available in the standard library of the programming language of your choice.

```
boolean isGarbage(String p, String m, int i)
```

Calling `isGarbage` with the source code for program text `p`, variable name `m`, and line number `i` has the following behavior.

`isGarbage(p, m, i)`	returns `true` if `m` is garbage at line `i` of program `p`.
`isGarbage(p, m, i)`	returns `false` otherwise.

You may assume that `isGarbage` always halts. You may also assume that `p` is "simple" code that does not contain class or function definitions.

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.