Homework 2

Due Sunday, September 25 by 10:00pm

Handout 6
CSCI 334: Fall 2022

Turn-In Instructions

For each question in this assignment, 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.

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`". 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.

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>/lab02.git`. For example, if your CS username is `22abc1`, the repository would be `https://evolene.cs.williams.edu/cs334-f22/22abc1/lab02.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, September 25 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)** "Advanced F#"
2. **(As needed)** Microsoft's Official F# Documentation

Problems

Q1. (15 points) ..... Peanut Butter and Jelly

`pbj(n: int) : string` is a function that returns a sentence, which is a `string` composed of a sequence of words. n is a positive, nonzero integer supplied by the user. *For each consecutive integer* between 1 and n inclusive, `pbj` either appends the empty string to its output or a word. A word is a string that contains the substring "`peanutbutter`" if n is evenly divisible by 3 or "`jelly`" if n is evenly divisible by 5. Each outputted word in the final sentence should be separated by a single space character. The entire sentence must end with the word "`time`" and a "`.`" character. Finally, use recursion to solve this problem.

Here are the first ten outputs of the program.

```
$ dotnet run 1
time.
$ dotnet run 2
time.
$ dotnet run 3
peanutbutter time.
$ dotnet run 4
peanutbutter time.
$ dotnet run 5
peanutbutter jelly time.
$ dotnet run 6
peanutbutter jelly peanutbutter time.
$ dotnet run 7
peanutbutter jelly peanutbutter time.
$ dotnet run 8
peanutbutter jelly peanutbutter time.
$ dotnet run 9
peanutbutter jelly peanutbutter peanutbutter time.
$ dotnet run 10
peanutbutter jelly peanutbutter peanutbutter jelly time.
```

The project directory for this question should be called "`q1`". You should be able to run your program on the command line by typing, for example, "`dotnet run 10`".

Q2. (10 points) ..... List duplication

Define a function `listDup(e: 'a)(n: int) : 'a list` that takes an element, e, of any type, and a non-negative number, n, and returns a list with n copies of e:

```
> listDup "moo" 4;;
val it : string list = ["moo"; "moo"; "moo"; "moo"]

> listDup 1 2;;
val it : int list = [1; 1]

> listDup (listDup "cow" 2) 2;;
val it : string list list = [["cow"; "cow"]; ["cow"; "cow"]]
```

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 moo 4`".

Q3. (30 points) ..... Zipping and Unzipping

(a) Write a function `zip(xs: 'a list)(ys: 'b list) : ('a * 'b) list` that computes the product of two lists of arbitrary length. You should use pattern matching to define this function:

```
> zip [1;3;5;7] ["a";"b";"c";"d"];;
val it : (int * string) list = [(1, "a"); (3, "b"); (5, "c"); (7, "d")]
```

When one list is longer than the other, repeatedly pair elements from the longer list with the last element of the shorter list.

```
> zip [1;3] ["a";"b";"c";"d"];;
val it : (int * string) list = [(1, "a"); (3, "b"); (3, "c"); (3, "d")]
```

In the event that one or both lists are completely empty, return the empty list. Note that in `dotnet fsi`, calling the function as below will produce an error because F# cannot determine the type of the element of an empty list.

```
> zip [1;3;5;7] [];;

  zip [1;3;5;7] [];;
  ^^^^^^^^^^^^^^^

code/stdin(14,1): error FS0030: Value restriction. The value 'it'
has been inferred to have generic type
    val it : ((int * int * int * int) * '_a) list
Either define 'it' as a simple data term, make it a function with
explicit arguments or, if you do not intend for it to be generic,
add a type annotation.
```

To make empty lists work, explicitly provide a type for the return value.

```
> let xs : (int * int) list = zip [1;3;5;7] [];;
val xs : (int * int) list = []
```

(b) Write the inverse function, `unzip(xs: ('a * 'b) list) : 'a list * 'b list`, which behaves as follows:

```
> unzip [(1,"a"); (3,"b") ;(5,"c"); (7,"de")];;
val it : int list * string list = ([1; 3; 5; 7], ["a"; "b"; "c"; "de"])
```

(c) Write `zip3(xs: 'a list)(ys: 'b list)(zs: 'c list) : ('a * 'b * 'c) list`, that zips three lists.

```
> zip3 [1;3;5;7] ["a";"b";"c";"de"] [1;2;3;4];;
val it : (int * string * int) list =
  [(1, "a", 1); (3, "b", 2); (5, "c", 3); (7, "de", 4)]
```

You must use `zip` in your definition of `zip3`.

(d) Provide a `main` function that exercises all of the above cases, plus a few more that you think of yourself.

The project directory for this question should be called "`q3`". You should be able to run this program using "`dotnet run`" without any additional arguments.

Q4. (20 points) ..... F# Map for Trees

(a) The binary tree datatype

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

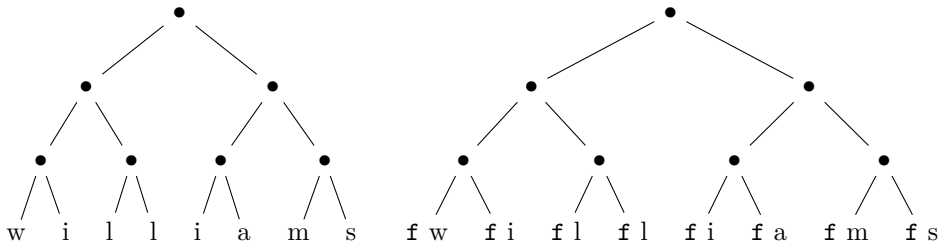
describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node).

Write the function

```
let 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 "`q4`". 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.

Q5. (20 points) ..... F# Reduce for Trees

The binary tree datatype

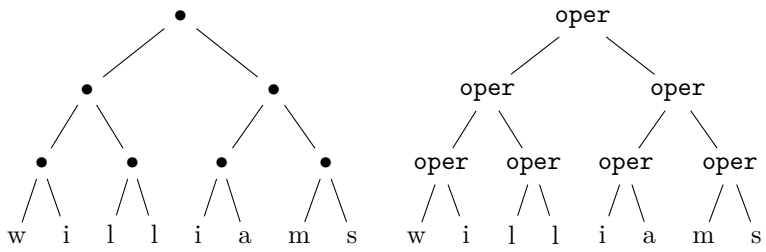
```
type Tree<'a> =
| Leaf of 'a
| Node of Tree<'a> * Tree<'a>
```

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 "`q5`". 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.

Q6. ($\frac{1}{10}^{\text{th}}$ 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.