

# Homework 7

Due Sunday, April 17 by 10:00pm

Handout 17  
CSCI 334: Spring 2022

---

## Turn-In Instructions

---

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334lab7_<your user name>`. For example, my repository would be `cs334lab7_dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

For this assignment, your completed submission should consist of two parts.

1. A  $\text{\LaTeX}$  source file called `lab7.tex` and pre-built PDF called `lab7.pdf` in a folder called `pset` that contains solutions to any written questions. A  $\text{\LaTeX}$  template is provided to help you get started. Be sure to `git add` all necessary files (e.g., images) if your  $\text{\LaTeX}$  depends on it. For full credit, please ensure that your  $\text{\LaTeX}$  file builds without error.
2. For each coding question in this assignment, create a project directory. For example, the source directory for question 1 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`”. Each program should be split into two pieces: a “`Program.fs`” file that contains the `main` method and associated program-startup routines (like argument parsers), 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.

---

## 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 (2 points).

This assignment is due on Sunday, April 17 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) “A Brief Introduction to F#”
2. (Required) “A Slightly Longer Introduction to F#”

---

## Problems

---

### Q1. (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 `fsharp`, 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.
- (e) Why can't you write a function `zip_any` that takes a list of any number of lists and zips them into tuples? From the first part of this question it should be clear that for any fixed  $n$ , one can write a function `zip $n$` . The difficulty here is to write a single function that works for all  $n$ . In other words, can we write a single function `zip_any` such that `zip_any [list1;list2;...;list $k$ ]` returns a list of  $k$ -tuples no matter what  $k$  is? Provide an answer in your `lab7.tex`  $\text{\LaTeX}$  file.

The project directory for this question should be called “q1”. You should be able to run this program using “dotnet run” without any additional arguments.

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

- (a) The binary tree datatype

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

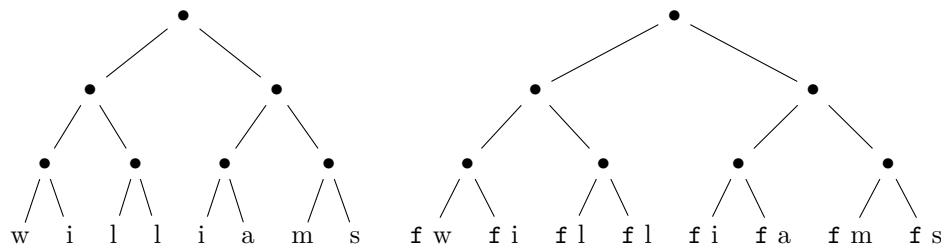
describes a binary tree for any type, but does not include the empty tree (i.e., each tree of this type must have at least a root node).

Write the function

```
let 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  $('a \rightarrow 'a) \rightarrow \text{Tree}'a \rightarrow \text{Tree}'a$ ? Provide an answer in your `lab7.tex` L<sup>A</sup>T<sub>E</sub>X 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” 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

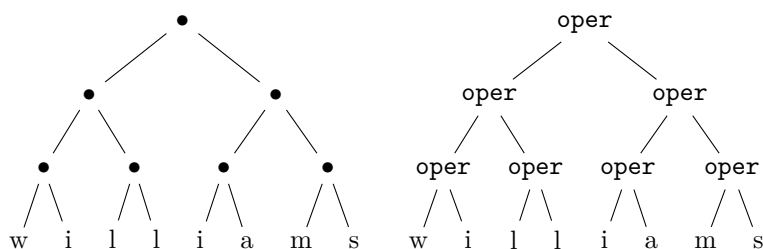
```
type Tree<'a> =
| Leaf of 'a
| Node of Tree<'a> * Tree<'a>
```

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 `maptree` 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. (10 points) ..... Five 5's

Consider the well-formed arithmetic expressions using the numeral 5. These are expressions formed by taking the integer literal 5, and the four binary arithmetic operators `+`, `-`, `*`, and `/`. Examples are 5, `(5 + 5)`, and `((5 + 5) * (5 - (5 / 5)))`. Such expressions correspond to binary trees where the internal nodes are operators and every leaf is a 5. You may recall that trees of this form are abstract syntax trees.

Note that in the examples above, we always surround subexpressions that use an arithmetic operator with parens to avoid ambiguous expressions.

Write an F# program that answers each of the following questions:

- What is the smallest positive integer that cannot be computed by an expression involving exactly five 5's?
- What is the largest prime number that can be computed by an expression involving exactly five 5's?
- Generate a human-readable expression (e.g., `((5 + 5) * (5 - (5 / 5)))`) that evaluates to that prime number.

You should start by defining an algebraic data type (i.e., a **type**) to represent arithmetic expressions involving the number 5 and the arithmetic operations. This question involves only integer arithmetic, so be sure to use **int** data.

Next, implement an **eval** function that takes an expression and computes the result. This evaluation function should call itself recursively until it hits a base case whose result is obvious. Think for a minute about what that base case might be.

You should then write a function that generates all possible expressions that contain a given number of 5's.

Answering the questions will involve an exhaustive search for all numbers that can be computed with a fixed number of 5's, so good programming techniques are important. Avoid unnecessary operations. You will also have to address the possibility of division by zero inside **eval**. There are a couple ways to do it— you could use an exception handler, or you could modify your **eval** function to work with an **option** type.

Be sure to document your code using (**\* F# comment blocks \***).

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 the program's output should clearly indicate the answers to the questions above.