Lab 6

Handout 17

Due Wednesday, March 19 by midnight

CSCI 334: Spring 2025

This review assignment requires that you write programs in F#, do parsing derivations, and write lambda calculus reduction proofs. Because you have done these things before, this assignment does not repeat instructions for doing those things. Refer back to previous assignments for instructions on creating F# console programs, and using the `ParseViz` and `LambdaViz` libraries.

As usual, each question asks you to provide specific functions. When asked to write console programs, ensure that your programs take input and provide output as directed. If your program takes inputs, always ensure that your program safely rejects bad inputs without crashing (i.e., it does "input validation").

Turn-In Instructions

Turn in your work using your assigned `git` repository. The name of your repository will have the form `https://aslan.barowy.net/cs334-s25/cs334-lab06-<USERNAME>.git`. For example, if your CS username is `abc1`, the repository would be `https://aslan.barowy.net/cs334-s25/cs334-lab06-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 Wednesday, March 19 by midnight.

Reading

1. All of the readings from this semester so far.

Problems

Q1. (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"]]
```

Your `listDup` function should be located in a file called `Library.fs` in a module called `CS334`. 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 moo 4`".

Q2. (10 points) ..... F# Reduce for Trees

The binary tree datatype

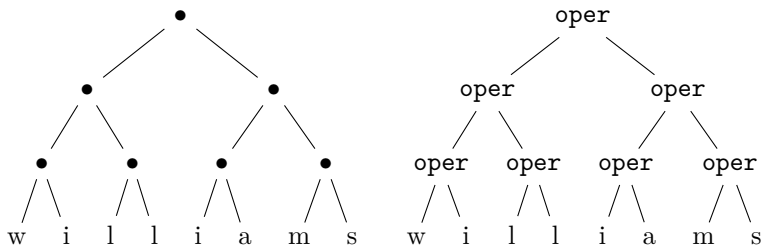
```
type Tree<'a> =
| Leaf of 'a
| Node of Tree<'a> * Tree<'a>
```

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.

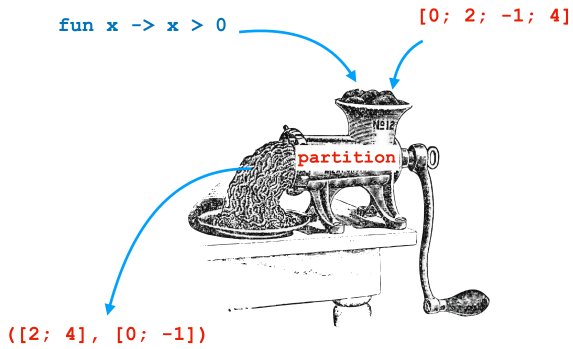
Your `treduce` function should be located in a file called `Library.fs` in a module called `CS334`. The project directory for this question should be called "`q2`". You should be able to run your program on the command line by typing, for example, "`dotnet run`" and output like the kind shown above should be printed to the screen. Be sure to provide several examples that demonstrate that your function works correctly.

Q3. (20 points) ..... Partition

In this question, you will write an F# function that partitions a list according to a predicate. The function, called `partition`, takes a predicate and a list as input. The function returns a 2-tuple. The left side of the 2-tuple contains the list of elements that match the predicate. The right side of the 2-tuple contains the list of all non-matching elements.

For example, the following use of `partition` puts positive numbers on the left and non-positive numbers on the right:

```
let xs = [0; 2; -1; 4]
let p = fun x -> x > 0
let (ys, zs) = partition p xs
```



The list `ys` contains

```
[2; 4]
```

and the list `zs` contains

```
[0; -1]
```

The `partition` function should start with the following declaration:

```
let rec partition (p: 'a -> bool)(xs: 'a list): 'a list * 'a list =
```

The logic of the partition function is essentially as follows:

- If the list is empty, return a pair of empty lists.
- If the first element of the list matches the predicate, prepend that element to the left list of the return value of the recursive call that partitions the rest of the list.
- If the first element does not match the predicate, prepend that element to the right list of the return value of the recursive call that partitions the rest of the list.

Your `partition` function should be located in a file called `Library.fs` in a module called `CS334`. 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. (15 points) ..... Parsing

Encode derivation trees for the following expressions. Use the following grammar. The start symbol is `<spell>`.

```
          <spell> ::= <simple>
                    | <enchanted>
                    | <extra-enchanted>
                    | <dangerous-words>
         <simple> ::= lumos
                    |  nox
                    |  alohamora
                    |  colloportus
      <enchanted> ::= <simple> maxima
                    |  amplio <simple>
                    |  <simple> augmenta
                    |  <simple> supra
<extra-enchanted> ::= omnara <enchanted> grandis
                    |  pleno <enchanted> exaltis
                    |  abra <enchanted> cadabra
<dangerous-words> ::= <spell> morelia volumnis
                    |  crescenda <spell> gigastra
```

(a) `lumos` in the function `q4a()`,

(b) `pleno colloportis maxima exaltis` in the function `q4b()`,

(c) `crescenda omnara amplio nox grandis morelia volumnis gigastra` in the function `q4c()`,

Use the `ParseViz` library to solve this problem. Each function above should be of type `unit -> Node`.

The project directory for this question should be called "`q4`". Your functions should be in a module called `CS334` in a file called `Library.fs` and your `main` function should be in a file called `Program.fs`.

Q5. (20 points) ..... Reduction

Reduce the following lambda expression to a normal form.

$$(\lambda x.\lambda y.xyy)(\lambda a.a)b$$

The following `Library.fs` function should return your proof.

```
let proof() : Proof list = ...
```

Use the `LambdaViz` library to solve this problem.

The project directory for this question should be called "`q5`". The `proof` function should be put in the file `Library.fs` in a module called `CS334`. Running your program on the command line by typing "`dotnet run`" should pretty-print your proof.

Q6. (20 points) ..... Reduction

Reduce the following lambda expression to a normal form.

$$(\lambda x.xx)(\lambda y.yx)z$$

The following `Library.fs` function should return your proof.

```
let proof() : Proof list = ...
```

Use the `LambdaViz` library to solve this problem.

The project directory for this question should be called "`q6`". The `proof` function should be put in the file `Library.fs` in a module called `CS334`.Running your program on the command line by typing "`dotnet run`" should pretty-print your proof.

Q7. ($\frac{1}{10}^{\text{th}}$ 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.