Report of Assignment 7

Haodong Liao & Liyuan Zhang

July 12, 2019

1 Preface

This is a LaTeXreport written by Haodong Liao and Liyuan Zhang, students of UESTC who participated in the summer school of UCPH. More codes of this summer school are at this repository.

2 Introduction

A string in F# is a sequence of characters[1], and some of the string functions are similar to list functions, after the practicing of the list function, it is time to move on.

In this assignment, we worked with simple text processing and met the requirements of analysing and generating some text according to statistics. In detail, we used pattern-matching and explicit type definition to create our own recursive functions, besides, we uesd powerful List-library functions to read, convert, analyze and generate target text.

3 Analysis and Design

3.1 Text Processing of character

3.1.1 ReadText

```
readText: filename:string -> string

// readFile.fsx
let filename = "readFile.fsx"
let line =
  try
    let reader = System.IO.File.OpenText filename
    reader.ReadToEnd ()
  with
    _ -> "" // The file cannot be read, so we return an empty string

printfn "%s" line
```

In this sub-assignment, we needed to convert *readFile.fsx* into a more flexible function which reads the content of any text file.

Referd to the input processing code in assignment 6, there were two branches needed to be handled, one was to set the entrypoint of our code, the other was to change the fixed argument to a flexible one so that the function could read the content of any text file. The key part of the pseudocode is shown as following:

```
[<EntryPoint>]
let main (input argument) =
  if (argument correct)
    call readText function with responding name of file
  else
    display error message
```

3.1.2 ConvertText

```
convertText: src:string -> string
```

[1] https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/lists

The goal of this sub-assignment was to convert letters of a string to lower case and removes all characters except a..z. According to [1], there was a *ToLower()* function *returns a copy of the string where each letter has been converted to lower case*, so we could use it to finish the 'convert' part and moved on to 'remove' part. The key part of the pseudocode is shown as following:

```
let convertText inputString =
  let lowerStr = inputString.ToLower()
    if (elment of lowerStr was in the range of a..z)
      keep the current element
    else remove the current element
```

3.1.3 Get Histogram

```
histogram: src:string -> int list
```

We needed to counts occurrences of each lower-case letter of the English alphabet in a string and returns a list in this sub-assignment. According to [1], there was a *String.filter* function similar to *List.filter* function, we used it to filter the specific letter of the string and counts the length of it to get the occurences. The key part of the pseudocode is shown as following:

```
let countchar (src:string)(letter:char) =
      if letter was in the range of a..z then
        filter the current letter
        handle the rest string
      else
        returned an empty list
let histogram (src:string): int list = (countchar src 'a')
```

3.1.4 Generate Ramdom String

```
randomString: hist: int list -> len:int -> string
```

The requirement of this sub-assignment was to change the given program and generate a string of a given length, and contains random characters distributed according to a given histogram which used our own functions. The flow of the program was shown as following:

```
step1 - Read the target text file and saved it
step2 - Converted the text to lower case & removed characters except a..z
step3 - Got the histogram of the text
step4 - Generated a new random string according to the histogram
```

3.2 Text Processing of pair of characters

3.2.1 Get Cooccurrence

In this sub-assignment, we needed to count occurences of each pairs of lower-case letter of the English alphabet in a string and return a list of lists. Our solution was to use double recursion and process the letter in each pair seperately. The key part of the pseudocode is shown as following:

```
// count the occurrence of each pair
let countnum =
  if the length of src > 2 then
    if the first two char = charpair then
      1 + sum recursively
    else
      ignore current and sum recursively
  else
    return 0

// handle the second letter
let countcharpair =
```

```
    if the second letter <= 'z' then
      sum the occurrence and process the left string recursively
    else return an empty list

  // process the first letter
  let countchar =
    if the first letter <= 'z' then
      process the second letter and process the left string recursively
    else return an empty list

  // call function from here
  let cooccurrence (src:string) = countchar src 'a'
```

3.2.2 Markov Model

```
fstOrderMarkovModel: cooc: int list list -> len:int -> string
```

We needed to generate a random string of length *len*, whose character pairs were distributed according to a specified cooccurrence histogram *cooc* of the story "Little Claus and Big Claus" in this sub-assignment. The flow of the program was shown as following:

```
step1 - Count the occurrence of each pairs (like 'aa' to 'az')
step2 - Count the coocurrence of all pairs ('aa' to 'zz')
step3 - Calculate the ratio of each pairs
step4 - Generate a random string of length len
  according to a given cooccurrence and the ratio of each pairs
```

4 Program description

Our implementation of **readText** function was as follows:

```
// 7.1
let printErrorMessage () =
    printfn "Program Input should be 'readfile filename'"

let readText (filename:string) : string =
    let line =
        try
          let reader = System.IO.File.OpenText filename
          reader.ReadToEnd ()
        with
          _ -> "" // The file cannot be read, so we return an empty string
    printfn "%s" line
    line

[<EntryPoint>]
let main (paramList : string[]) : int =
    if paramList.Length <> 2 then
        printErrorMessage ()
        0
    else
        match paramList.[0] with
            | "readfile" ->
                let str = readText paramList.[1]
                printfn "str : %A" str
            | _ ->
                printErrorMessage ()
        1
```

As it shows, the key of this function was to match the argument list, the first argument should be the command 'readfile', and the second argument was the name of file.

Implementation of **convertText** function was as follows:

```
// 7.2
let rec convertText (src:string) : string =
    let lowerSRC = src.ToLower()
    match lowerSRC with
    | "" -> ""
    | elm ->
        if ('a' <= elm.[0] && elm.[0] <= 'z')
        then
            elm.[0].ToString() + (convertText elm.[1..])
        else
            convertText elm.[1..]
```

It made us wondering whether there was an expression of string could work as 'elm::rest' to represent a list at the beginning, things did not go well so we changed our mind to use elm[1..] to represent the rest of a string.

Implementation of **histogram** function was as follows:

```
// 7.3
let rec countchar (src:string)(letter:char): int list =
  if letter >= 'a' && letter <= 'z' then
    ((String.filter (fun char -> char=letter) src).Length)::
     (countchar src (char (int letter + 1)))
  else []
let histogram (src:string): int list = (countchar src 'a')
```

This function was the key to solving the following problems.

Assignment 7.4 was a summary of the previous three assignments, and the key part of implementation of it was as follows:

```
/// step1 readfile
let str = readText "littleClausAndBigClaus.txt"
/// step2 convertfile
let convertRes = convertText str
/// step3 get histogram
let hist = histogram convertRes
let alphabet = List.init hist.Length (fun i -> 'a' + char i)
printfn "A histogram:\n %A" (List.zip alphabet hist)
/// 7.4 step4 generages a new random string
let ranStr = randomString hist convertRes.Length
printfn "A random string: %s" ranStr
let newHist = histogram ranStr
printfn "Resulting histogram:\n %A" (List.zip alphabet newHist)
```

It was important to have a clear thought of processing flow.

Implementation of **cooccurrence** function was as follows:

```
// 7.5
let rec countnum (src:string) (charpair:string):int =
  if src.Length >=2 then
    if src.[0..1] = charpair then
      1+(countnum src.[1..] charpair)
    else
      (countnum src.[1..] charpair)
  else
    0
let rec countchapair (src:string) (letter1:char) (letter2:char):int list =
  if letter2 <= 'z' then
    let charpair = string letter1 + string letter2
    (countnum src charpair)::(countcharpair src letter1 (letter2+char 1))
  else []

let rec countchar (src:string) (letter1:char):int list list =
  if letter1 <= 'z' then
    (countcharpair src letter1 'a')::(countchar src (letter1 + char 1))
  else []
```

```
let cooccurrence (src:string):int list list =
  countchar src 'a'
```

This was a difficult problem for us, and the key to the solusion was the idea of 'divide and conquer', if we could not process the pair of letters at a time, we handle the letter of it separately.

Assignment 7.6 was a summary of all the previous assignments, and the key part of implementation of it was as follows:

```
let fstOrderMarkovModel (cooc:int list list) (len:int):string =
    let strLengthlist =
      List.map (fun x -> (float x)) (List.map (List.sum) cooc)
    let strLengthSum = List.sum strLengthlist
    let ratiolist = List.map (fun x -> x/strLengthSum) strLengthlist
    let randomStringAlt (hist: int list):string = randomString hist
      (int((float (List.sum (hist))/strLengthSum)*(float len)))
    let strlist = List.map (randomStringAlt) cooc
    let composestr (acc:string) (elm:string):string = acc + elm
    List.fold composestr "" strlist
printfn "test: \n %A" (fstOrderMarkovModel (cooccurrence a) (a.Length))
printfn "present cooc : \n %A"
  (cooccurrence (fstOrderMarkovModel (cooccurrence a) (a.Length)))
```

This was another difficult assignment for us, the main challenge was to arrange the text properly so the occurrence was similar. We tried to generate a new character according to the letter before the current letter, but things just did not worked well for reason we did not figured out, then we tried with the weight of the pair based on the proportion of the sum of the single letters to the sum of the occurrences of all the letters, for example: $Weight_{'a'} = \frac{Occurrenceof'aa'to'az'}{Occurrenceof'aa'to'zz'}$. In this way, when there was a given length, the length of each character list (like 'aa'..'az') would be as the weight of all character list, so that we could generate a random string which had a similar distribution but different content.

5 Evaluation

The testing environment was macOS Mojave 10.14.5 system with iTerm and Microsoft (R) F# Compiler version 4.1.

We combined all the assignment into a single program, complied and tested it with parameter *readfile littleClausAndBigClaus.txt*, and the key part of result (because it was to long) were showed as following:

```
→ fsharpc Assig2.fsx && mono Assig2.exe readfile littleClausAndBigClaus.txt
Microsoft (R) F# Compiler version 4.1
Copyright (c) Microsoft Corporation. All Rights Reserved.
str :
In a village there lived two men who had the self-same name. Both were named Claus. But one of t
hem owned four horses, and the other owned only one horse; so to distinguish between them people
 called the man who had four horses Big Claus, and the man who had only one horse Little Claus.
Now I'll tell you what happened to these two, for this is a true story.
```

Figure 1: Testing of readText function

```
convertedstr :
"inavillagetherelivedtwomenwhohadtheselfsamenamebothwerenamedclausbutoneofthemownedfourhorsesand
theotherownedonlyonehorsesotodistinguishbetweenthempeoplecalledthemanwhohadfourhorsesbigclausand
themanwhohadonlyonehorselittleclausnowilltellyouwhathappenedtothesetwoforthisisatruestorythewhol
eweekthroughlittleclaushadtoplowforbigclausandlendhimhisonlyhorseinreturnbigclauslenthimallfouro
fhishorsesbutonlyforonedayaweekandthathadtobesundayeachsundayhowproudlylittleclauscrackedhiswhip
overallthefivehorseswhichwereasgoodashisownonthatdayhowbrightlythesunshonehowmerrywerethechurchb
ellsthatranginthesteeplehowwelldressedwereallthepeoplewhopassedhimwithhymnbookstuckedundertheira
rmsandastheywenttheirwaytochurchtohearttheparsonpreachhowthepeopledidstaretoseelittleclausplowing
withallfivehorsesthismadehimfeelsoproudthathewouldcrackhiswhipandhollogetupallmyhorsesyoumustnot
```

Figure 2: Testing of convertText function

6 Conclusion

In this assignment, I implemented the *myFold* and *myFilter* functions with my own recursive method. I stuck at the beginning of my writing, but things went smoothly when I rearranged my thoughts and wrote down the pseudocode. It's

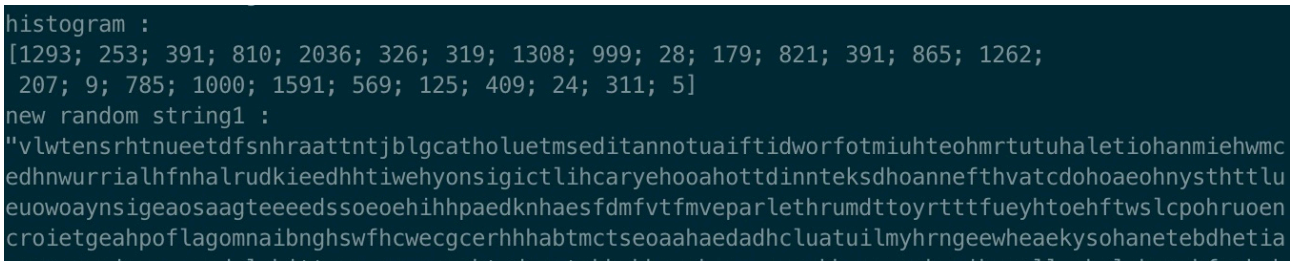

Figure 3: Testing of histogram and randomString function

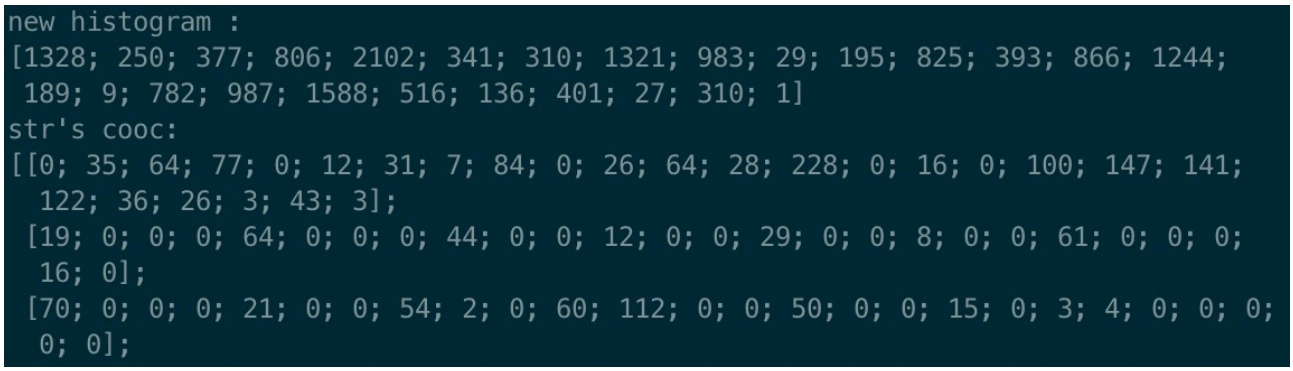

Figure 4: Histogram and Cooc of Generate String of randomString function

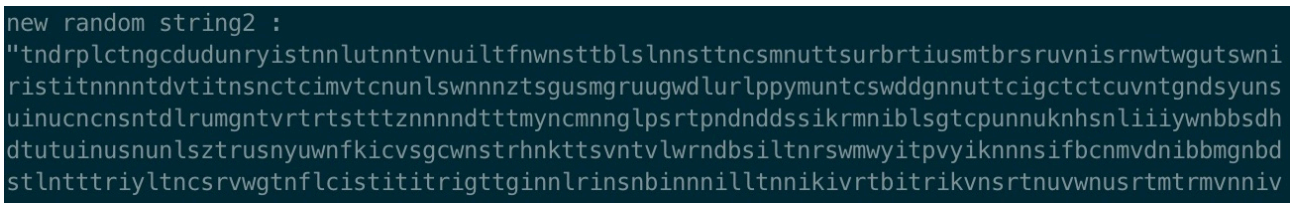

Figure 5: Testing of fstOrderMarkovModel function

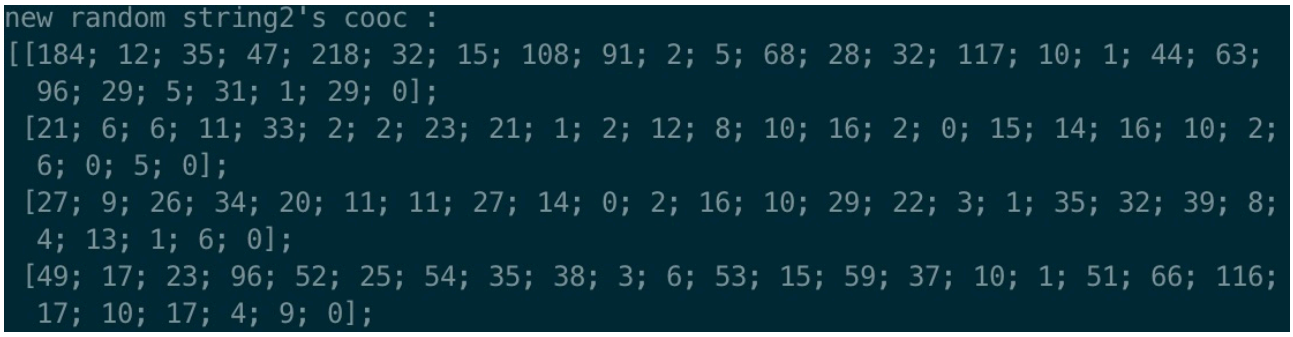

Figure 6: Coocurrence of fstOrderMarkovModel function

easy to get bogged down in the details of a program, but we should take a top-down functional programming approach and thinking more about what to do than how to do it.

References

[1] Jon Sporring. *Learning to Program with F#.* 2019.