Declarative Programming Coursework

Pandelis Zembashis, S15101590

July 31, 2017

1 Pangram

1.1 Brief

Write a program which takes in a path to an input file, reads that file and then divides it into a set of elements representing each sentence within the file. Each sentence within the file should then be checked as to whether it is a Pangram.

A pangram is a sentence that contains all the letters of the English alphabet at least once.

1.2 Planning and Design

1.2.1 Defining behaviour

We will once again be working on a utility program in which we pass a file to the utility and the utility will tell us about the file. We want to know how many pangrams are found in the file if at all, and what they are. Ideally we will also say where they were found in the file.

1.2.2 Approach

We will also be utilising a test driven development method with this utility in order to know when we have achieved the features and usability we intend to have.

The definition of a pangram is "all letters of the English alphabet at least once". **At least once** leads me to believe that we can utilise the unique nature of Sets to our advantage. Sets can be compared with each other and only contain a value once. So if we can convert all our sentences to sets and compare them with a base alphabet set, we can determine if it is a pangram.

1.2.3 Technical Requirements

After learning more about NUnit working on the previous application we will be adopting a slightly different project structure this time round.

So that we can package up a binary with nothing but the code to run and to be able to test all files including Main.fs, we will split this solution file into two project.

Pangram.fsproj and Pangram.Tests.fsproj

The files in Pangram will be (in execution order):

1. File.fs - File IO operations
2. Pangram.fs - Pangram checking
3. Main.fs - Entry point where File meets Pangram function and feedback to user

The files in Pangram.Tests will be:

1. Tests.fs - Unit tests

1.3 Implementation

The full source code is available in the appendix. Please note that unit tests will be shown inline with their accompanying function where applicable. In reality all tests exist in Tests.fs

Please also note that double single quotes (") are used in test function names as there were problems displaying them as double back ticks (`)

1.3.1 Pangram.fs

Because we are doing similar IO and text operations as the previous project we can reuse and adapt some of those well tested and reliable functions in this application. That is the benefit of writing small reusable and testable chunks.

We can also use some of what we learnt to write better pipelines and cleaner functions.

```
module Pangram
module Alphabet
    onlyLetters: char -> bool
    allLower: char -> char
    explode: string -> Set<char>

matchToString: Match -> string
sentences: string -> seq<string>
alphabet: Set<char>
isPangram: string -> bool
findPangrams: seq<string> -> seq<string>
Find: string -> seq<string>
```

We have kept the Alphabet module and adjusted it to our needs. However this time rather than organising into another module for the rest of the functions, the Pangram module scope will suffice.

module Alphabet

```
let onlyLetters = fun l -> System.Char.IsLetter(l)

let allLower = fun l -> System.Char.ToLower(l)

let explode (s:string) =
    set[for char in s -> char]
    |> Set.filter onlyLetters
    |> Set.map allLower

[<Test>]
let ''String to lowercase letter only set''() =
    let i = "HeL27&@lo00"
```

```
    let o = set['h';'e';'l';'l';'o']
    Assert.AreEqual(o, Pangram.Alphabet.explode i)

[<Test>]
let ''Set equality''() =
    let i = Pangram.Alphabet.explode "
        ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
        "
    let o = set['a'..'z']
    Assert.AreEqual(o, i)

[<Test>]
let ''Set inequality''() =
    let i = Pangram.Alphabet.explode "abhoisud"
    let o = set['a'..'z']
    Assert.AreNotSame(o, i)

[<Test>]
let ''Set equality even with random chars''() =
    let i = Pangram.Alphabet.explode "
        ABCDEFGHIJKLMNO7638&*62786187yS&*Dg78@198@873
        **@37PQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
    let o = set['a'..'z']
    Assert.AreEqual(o, i)
```

onlyLetters and **allLower** are helpers for a Filter that we will carry out during exploding of text.

Our explode function this time round does an extra step. It uses the higher order Filter and Map to get our set into the right format for comparison with the alphabet.

module Pangram

```
let matchToString (m : Match) = m.Value

let sentences (s:string) =
    //Sentence regex
    let re = new Regex("[^.!?\s
        ][^.!?]*(?:[.!?](?!['\"]?\s|$)[^.!?]*)
        *[.!?]?['\"]?(?=\s|$)")
    re.Matches(s)
    |> Seq.cast
    |> Seq.map matchToString

[<Test>]
let ''Convert string of sentences to list''() =
```

```
let incoming = "Edit the Expression & Text to
    see matches. Roll over matches or the
    expression for details. Undo mistakes with
    cmd-z. Save Favorites & Share expressions
    with friends or the Community. Explore your
    results with Tools. A full Reference & Help
    is available in the Library, or watch the
    video Tutorial."
let list = ["Edit the Expression & Text to see
    matches.";"Roll over matches or the
    expression for details."; "Undo mistakes with
     cmd-z."; "Save Favorites & Share expressions
     with friends or the Community."; "Explore
    your results with Tools."; "A full Reference
    & Help is available in the Library, or watch
    the video Tutorial."]
Assert.AreEqual(list, Pangram.sentences incoming
    )
```

In order to get a Sequence of sentences out of our file we use a very elaborate Regex. The regex allows for the following

- `[^.!?\\s]}` - First character is neither punctuation not whitespace
- `[^.!?]*}` - Match anything up to punctuation
- `[.!?] (?!['\"]?\\s|$)` - Match inner punctuation if not followed by whitespace of end of stream
- `[.!?]?` - Optionally end with punctuation
- `['\"]?` - Match quotations with inner punctuation if that applies

The **Regex.Matches** class returns a **System.Text.RegularExpressions.MatchCollection** which we can cast to a sequence and iterate over with a map. The map gets the underlying value from the regex class. The matched sentence. We now have a sequence of sentences that appear in the string.

```
let alphabet = set['a' .. 'z']
let isPangram =
    fun s -> alphabet.Equals( Alphabet.explode s )
```

Utilizing the unique nature of Sets isPangram takes a set and checks if it matches the alphabet set that we defined. If it matches, then we must have a pangram.

```
let findPangrams ls =
    ls |> Seq.filter isPangram
```

```
let Find (s:string) =
    s |> sentences |> findPangrams
    [<Test>]
let ''Pangram detection are same''() =
    let list = set["Edit the Expression & Text to
       see matches.";"Roll over matches or the
       expression for details."; "Undo mistakes with
        cmd-z."; "Save Favorites & Share expressions
        with friends or the Community."; "Explore
       your results with Tools."; "the quick brown
       fox jumps over the lazy dog."; "A full
       Reference & Help is available in the Library,
        or watch the video Tutorial."]
    let o = set["the quick brown fox jumps over the
       lazy dog."]
    Assert.AreEqual(o, Pangram.findPangrams list)

[<Test>]
let ''Multiple pangram detection''() =
    let list = set["Edit the Expression & Text to
       see matches."; "This Pangram contains four
       a s, one b, two c s, one d, thirty e s,
        six f s, five g s, seven h s, eleven
       i s, one j, one k, two l s, two m s,
       eighteen n s, fifteen o s, two p s, one
        q, five r s, twenty-seven s s, eighteen
       t s, two u s, seven v s, eight w s,
       two x s, three y s, & one z.";"Roll over
       matches or the expression for details."; "
       Undo mistakes with cmd-z."; "Save Favorites &
        Share expressions with friends or the
       Community."; "Explore your results with Tools
       ."; "the quick brown fox jumps over the lazy
       dog."; "A full Reference & Help is available
       in the Library, or watch the video Tutorial."
       ]
    let o = set["the quick brown fox jumps over the
       lazy dog."; "This Pangram contains four a s
       , one b, two c s, one d, thirty e s, six
       f s, five g s, seven h s, eleven i s,
        one j, one k, two l s, two m s, eighteen
        n s, fifteen o s, two p s, one q, five
        r s, twenty-seven s s, eighteen t s,
       two u s, seven v s, eight w s, two
        x s, three y s, & one z."]
    Assert.AreEqual(o, Pangram.findPangrams list)
```

```
[<Test>]
let ''Pangram detection fails''() =
    let list = set["Edit the Expression & Text to
       see matches."; "This Pangram contains four
       a  s , one b, two  c  s , one d, thirty  e  s ,
        six  f  s , five  g  s , seven  h  s , eleven
       i  s , one j, one k, two  l  s , two  m  s ,
       eighteen  n  s , fifteen  o  s , two  p  s , one
        q, five  r  s , twenty-seven  s  s , eighteen
       t  s , two  u  s , seven  v  s , eight  w  s ,
       two  x  s , three  y  s , & one z.";"Roll over
       matches or the expression for details."; "
       Undo mistakes with cmd-z."; "Save Favorites &
        Share expressions with friends or the
       Community."; "Explore your results with Tools
       ."; "the quick brown fox jumps over the lazy
       dog."; "A full Reference & Help is available
       in the Library, or watch the video Tutorial."
       ]
    let o = set["the quick brown fox jumps over the
       lazy dog."]
    Assert.AreNotEqual(o, Pangram.findPangrams list)

[<Test>]
let''Find in string''() =
    let i ="Lots of sentences. THat arent right. the
        quick brown fox jumps over the lazy dog.
       lalallala"
    let o = set["the quick brown fox jumps over the
       lazy dog."]
    Assert.AreEqual(o, Pangram.Find i)
```

Finally we can put this all together with our final two functions. The Find method is the only function we will need to call externally and all the plumbing has been setup. When we pass a string of the document we want to find pangrams into Find, it will break it down into a Squence of sentences, filter out anything that doesnt match against the alphabet and we will be left with a sequence of strings that will be the pangrams found in the text.

1.4 File.fs

```
module FileIO
Open: string -> string
```

module FileIO

```
let Open(path:string) =
    match File.Exists(path) with
    | true ->
        File.ReadAllText(path)
    | _ -> invalidArg "file" "The file specified
       does not exist"
```

Instead of reading all the lines to a list instead we take in all the file as a string so that we can run the regex against it.

1.5 Main.fs

We simplified here heavily since our last project. Main now only does user interface and feedback to the user as we have taken care of all the plumbing in our logic portion.

```
[<EntryPoint>]
main: string[] -> int


let main argv =
    if (argv.Length = 0) then
          printfn("Please provide the route to a
             file on the system to process")
          1 //No file found so return an error exit
             code
    else
      let foundPangrams = FileIO.Open argv.[0] |>
         Pangram.Find
      let numberFound = foundPangrams |> Seq.length


      let printFound n =
          if (n = 0) then
            printfn "No pangrams found in %s" argv
               .[0]
            exit 0 //exit the program
          else if (n = 1) then
            printfn "Found 1 pangram"
          else
            printfn "Found %i pangrams" n

      printFound numberFound

      //List the found pangrams to stdout
      let showPangrams found =
        found
        |> List.ofSeq
```

```
        |> List.map(fun line -> printfn "%s\n" line
           )
        |> ignore


    showPangrams foundPangrams

    0
```

Once again as we call the program we will need to give feedback to the user if he has not supplied a valid argument or the location to the file does not exist.

Once we have opened the file however we can pipe it to our Pangram.Find function to get out the pangrams.

We then print out one of three options to the screen, if no pangrams were found we will let the user know. Otherwise we list how many followed by each pangram printed to their screen with a new line seperating each result.

I had to cast to a List in showPangrams because Seq.map(fun line -¿ printfn ”

Program then exits with a code 0