Test Assignment

Ib Scherer - kcp642

assignment description

1 Reflection on group work and sources (20 points)

1.1 a

Who are the members of your group that participated in the assignment?

The group comprised of Mikkel Nielsen (gcs767), Sebastian Rasmussen (hdn188), Simon Svarre (hrk999), Mikkel Vogelius (tzm978), and myself (kcp642).

1.2 b

Describe how you organized group work for the assignment, and reflect on the quality of the group interactions. (Hint: You can rely on Irina's video to guide your reflection)

All members where present on Monday the 20th where we held a meeting about how we would go about completing the assignment. At the meeting we agreed on having a shared testing suite for the Pascal exercise as well as a shared framework for the code, compromising of all the files and headers for each function.

The Pascal test-suite was constructed on 20th and shared via GitHub. We also agreed on using the GitHub as a workspace for our own code, branching the main shared code base into individual branches to write our own solutions. This was done so that input on problems could be dealt with remotely.

Once the structure was set, most of the work was individual and we stayed in contact via Messenger. I was not able to make it for our Thursdays meeting the 23rd, but the time was set to meet as a group and discuss the assignment.

1.3 c

Give a list of the external sources you used during the assignment, and explain how you used them.

Extensive list of GenAI use and sources can be found in the Log.txt file. My primary source for information was Wikipedia (link: https://en.wikipedia.org/wiki/Pascal%27s_triangle) and GeeksForGeeks (https://www.geeksforgeeks.org/maths/pascals-triangle-formula/) for understanding the concept of Pascal's Triangle further.

The List Module from the FSharpCorewas GitHub was also referenced in regards to solving the MyList part of the assignment. (link: https://fsharp.github.io/fsharp-core-docs/reference/fsharp-collections-listmodule.html)

PowerPoints from previous lectures and notes taken from previous lectures were also used in solving the assignment.

2 Pascal's triangle (45 points)

2.1 a

Define a recursive function that given n and k as an input pair, calculates the corresponding binomial coefficient.

Specification: Takes a pair of natural numbers n and k, and returns the corresponding binomial coefficient for n choose k
Function name: pascal
input: int * int
output: int

```
let rec pascal (n: int, k: int) : int =
    if k <= 0 && n <= 0 && n >= k then
        match n, k with
        | _, 0 -> 1
        | n, k when n = k -> 1
        | _ -> pascal (n - 1, k - 1) + pascal (n - 1, k)
    else
        0
```

2.2 b

Define an imperative function (using for or while loops instead of recursion) that calculates the same thing as (a)

Specification: Takes a pair of natural numbers n and k, and returns the corresponding binomial coefficient for n choose k
Function name: pascal
input: int * int
output: int

when implementing I used the recommended mutable int list list data type for keeping track of the constructed triangle, but due to hardware limitations when running the testing file found a int array array to be more useful. With the testing also originally using the max int value, I decided on constructing only half the triangle to save on memory - note that solving for the center of the triangle results in relatively messy code, but twice as efficient memory wise.

```
    let pascalNoRec (n: int, k: int) : int =
    // check for natual numberso and than n >= k
    if n < k || k < 0 then
        0
    // check for edges
    else if n = k || k = 0 then
        1
    else
        // initialize matrix
        let matrix : int array array = Array.zeroCreate (n + 1)
        for i = 0 to n do
            // each row i has (i/2)+1 elements, since pascal's
               triangle is symmetric, and we use the mirroring
               property to calculate results.
            matrix.[i] <- Array.create ((i/2)+1) 1

        // fill in the matrix using pascal's rule, skipping the
           first 2 rows and the left edges
        for i = 2 to n do
            for j = 1 to matrix[i].Length - 1 do
```

```
18                  // check for right edge (middle element
19                  if j = matrix[i].Length - 1 then
20                      // handle middle element with mirroring property
21                      if i % 2 = 0 then
22                          matrix.[i].[j] <- matrix.[i - 1].[j - 1] * 2
23                      else
24                          matrix.[i].[j] <- matrix.[i - 1].[j] +
                               matrix.[i - 1].[j - 1]
25                  // normal case
26                  else // pascal's rule
27                      matrix.[i].[j] <- matrix.[i - 1].[j - 1] +
                           matrix.[i - 1].[j]
```

2.3 c

Pascal's triangle is defined for natural numbers where $n \geq k$ What is a good value to return when this condition doesn't hold?

To keep it simple, this implementation returns the value 0 when a faulty input is given. This value does not exist in the triangle, which makes it unique compared to any other non-negative number. We could also have used -1, for similar reasons, but prefer using 0 as it also equates to the boolean 'FALSE'.

2.4 d + e + f

Design a test suite to systematically test input for pascal and pascalNoRec The test-suite was designed as a group to test both properties for both the non-recursive and the recursive implementation of the Pascal function. Please reference source code for the code, found under /src/Pascal/PascalTest.fsx.

When testing I had issues with memory leaks and int overflows for my original solutions (original solution was using a factorial helper function called pascalFactCalc, included in source code but unused). Therefore I updated the values of n and max to be smaller and more easily testable.

the tests test edge-cases and base cases for both the recursive and the non-recursive implementation, for both property 1 and property 2 of Pascal's Triangle. I also notified the group that the values being tested where a bit extreme on the 24th, as that was when I finished my code. I changed the max value to be 33 and the n value to be 17, as the 33rd row in pascals triangle is the limit for how much an int can represent.

Currently the tests do not test whether or not the recursive an the non-recursive yield the same results. This could be added as further testing.

When running the test suite all results return positive on my computer.

3 Implementing helper functions for lists (35 points)

3.1 a + b

Write the specifications for take, drop, length and map in a file MyList.fsi.

The implementations follow the specifications and definitions as specified in the assignment. See source cdoe for commented version of the code.

```
1  type 'a MyList =
2      | Empty
3      | Something of 'a * 'a MyList
4
5  let rec take (n: int) (lst: 'a MyList) : 'a MyList =
6      match lst with
7      | Empty -> Empty
8      | Something (head, tail) ->
```

```
9           if n <= 0 then
10              Empty
11          else
12              Something (head, take (n - 1) tail)
13
14  let rec drop (n: int) (lst : 'a MyList) : 'a MyList =
15      match lst with
16      | Empty -> Empty
17      | Something (head, tail) ->
18          if n <= 0  then lst
19          else drop (n - 1) tail
20
21  let rec length (lst: 'a MyList) : int =
22      match lst with
23      | Empty -> 0
24      | Something (head, tail) -> 1 + length tail
25
26
27  let rec map (f: 'a -> 'b) (lst: 'a MyList) : 'b MyList =
28      match lst with
29      | Empty -> Empty
30      | Something (head, tail) -> Something (f head, map f tail)
```

3.2 c

Write code to try out each of the functions in a file called MyListExamples.fsx.

Examples are written and can be viewed in the source file. Examples ensure that MyLists can both be of integers and strings. For every helper function there are also 1-3 examples using either the int MyList and string MyList.

3.3 d + e

Are there inputs your functions don't work for? Make sure these cases are included in their specification, or that their behavior in those cases are explained. Argue for why your solution makes sense.

Due to time limitation I have not extensively tested my own code to find any inputs that do not work. The arbitrary input type takes any input the same way as the FSharpCore List takes inputs, but it is less efficient and takes more keystrokes to use. I see little use of the MyList function as it exists currently, but with hyper specialized needs perhaps there are uses for your own definition of a list.

the project file was made in collaboration with the rest of group 18 and runs without issues when using the dotnet run command.