Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 3 - individuel opgave

Jon Sporring

23. september - 1. oktober.
Afleveringsfrist: lørdag d. 1. oktober kl. 22:00.

I denne periode skal vi arbejde med lister. Lister er den første af en række abstrakte datastrukturer, vi skal kigge på. Lister er så vigtig en datastruktur at F# både har syntaks der direkte understøtter dem og et bibliotek (modul), `List`, med mange ekstra funktioner til listebearbejdning.

Denne arbejdsseddels læringsmål er:

- at kunne arbejde med anonyme funktioner,
- at kunne oprette, gennemløbe og lave beregninger med lists vha. `List`-modulet,
- at kunne skrive rekursive funktioner, som tager lister som argument og som giver lister som returværdi,

Opgaverne er opdelt i øve- og afleveringsopgaver. I denne periode skal I arbejde individuelt med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i "'Noter, links, software m.m."'→"'Generel information om opgaver"'.

Øveopgaver (in English)

3ø0 In the following, you are to work with different ways to create a list:

(a) Make an empty list, and bind it with the name `lst`.

(b) Create a second list `lst2`, which prepends the string `"F#"` to `lst` using the cons operator `::`. Consider whether the types of the old and new list are the same.

(c) Create a third list `lst3` which consists of 3 identical elements `"Hello"`, and which is created with `List.init` and the anonymous function `fun i -> "Hello"`.

(d) Create a fourth list `lst4` which is a concatenation of `lst2` and `lst3` using "`@`".

(e) Create a fifth list `lst5` as `[1; 2; 3]` using `List.init`

(f) Write a recursive function `oneToN : n:int -> int list` which uses the concatenation operator, "`@`", and returns the list of integers `[1; 2; ...; n]`. Consider whether it would be easy to create this list using the "`::`" operator.

(g) Write a recursive function `oneToNRev : n:int -> int list` which uses the cons operator, "`::`", and returns the list of integers `[n; ...; 2; 1]`. Consider whether it would be easy to create this list using the "`@`" operator.

3ø1 Use `List.map` write a function, which takes a list of integers and returns the list of floats where each element has been divided by `2.0`. For example, if the function is given the input `[1; 2; 3]`, then it should return `[0.5; 1.0; 1.5]`.

3ø2 Write a recursive function `rev: 'a list -> 'a list`, which uses the concatenation operator `"@"` to reverse the elements in a list.

3ø3 Write the types for the functions `List.filter` and `List.foldBack`.

3ø4 Make a function `avg: (lst: float list) -> float` using `List.fold` and `lst.Length` which calculates the average value of the elements of `lst`.

Afleveringsopgaver (in English)

In the following we are going to work with lists and Canvas. The module Canvas has the ability to perform simple turtle graphics. To draw in turtle graphics, we command a little invisible turtle, which moves on the canvas with a pen. The function `turtleDraw` is given a list of `turtleCmds`, such as `PenUp` and `PenDown` to raise and lower the pen, `Turn 250` and `Move 100` to turn 250 degrees and move 100 pixels, and `SetColor red` to pick a red pen. For example in Figure 1, the function `tree sz` creates the set of turtle commands for drawing a fractal tree of size `sz` and returns the turtle to the starting point. The command `turtleDraw` executes the list of turtle commands, which in this case draws the tree on a canvas and displays it. In this exercise, you are to work with turtle commands.

3i0 Consider a list of pairs of integers `(dir,dist)` which are to be translated into a list of turtle commands, such that `[(10, 30); (-5, 127); (20, 90)]` is translated into `[Turn 10; Move 30; Turn -5; Move 127; Turn 20; Move 90]`. You are to

(a) Write the following functions:

```
type move = int*int // a pair of turn and move
fromMoveRec: lst: move list -> Canvas.turtleCmd list
fromMoveMap: lst: move list -> Canvas.turtleCmd list
fromMoveFold: lst: move list -> Canvas.turtleCmd list
fromMoveFoldBack: lst: move list -> Canvas.turtleCmd list
```

where

i. `fromMoveRec` is a recursive function, that does not use the List module

ii. `fromMoveMap` is non-recursive function, which uses `List.map`

iii. `fromMoveFold` is non-recursive function, which uses `List.fold`

iv. `fromMoveFoldBack` is non-recursive function, which uses `List.foldBack`

In some of the above functions, you may also find it useful to use `List.concat` and `List.rev`.

```
#r "nuget:diku.canvas, 1.0.1"
open Canvas

let rec tree (sz: int) : Canvas.turtleCmd list =
   if sz < 5 then
     [Move sz; PenUp; Move (-sz); PenDown]
   else
     [Move (sz/3); Turn -30]
     @ tree (sz*2/3)
     @ [Turn 30; Move (sz/6); Turn 25]
     @ tree (sz/2)
     @ [Turn -25; Move (sz/3); Turn 25]
     @ tree (sz/2)
     @ [Turn -25; Move (sz/6)]
     @ [PenUp; Move (-sz/3); Move (-sz/6); Move (-sz/3)]
     @ [Move (-sz/6); PenDown]

let w = 600
let h = w
let sz = 100
turtleDraw (w,h) "Tree" (tree sz)
```

Figure 1: A turtle graphics program for drawing a fractal tree in Canvas.

(b) Demonstrate that these 4 functions produce the same result given identical input.

3i1 (a) The following program

```
let rnd = System.Random()
let v = rnd.Next 10
```

makes a random integer between the integer $0 \leq v < 10$. Use this to make a function

`randomTree: sz: int -> Canvas.turtleCmd list`

which calls `tree sz` and concatenates further turtle commands to place a tree randomly on a canvas, and return the turtle to the origin. Test your function by calling `turtleDraw` with such a list.

(b) Write a recursive function

`forest: sz: int -> n: int -> Canvas.turtleCmd`

which makes n random trees on the canvas by calling `randomTree` n times. Test your function by calling `turtleDraw` with such a list.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `3i.zip`
- en opgavebesvarelse i pdf-format.

Zip-filen skal indeholde:

- filen `README.txt` som er en textfil med jeres navn og dato arbejdet.
- en `src` mappe med følgende og kun følgende filer:

  3i0a.fsx, 3i0b.fsx, 3i1a.fsx og 3i1b.fsx

  svarende til afleveringsopgaverne. Funktionerne skal være dokumenteret med ifølge dokumentationsstandarden ved brug af `<summary>`, `<param>` og `<returns>` XML tagsne.
- pdf-dokumentet skal være lavet med LaTeX, benytte `opgave.tex` skabelonen, ganske kort dokumentere din løsning og indeholde figurer, der viser outputgrafik fra canvas for opgaverne.

God fornøjelse.