Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 5 - gruppeopgave

Jon Sporring

28. september - 3. oktober.
Afleveringsfrist: lørdag d. 3. oktober kl. 22:00.

Med denne arbejdsseddel starter vi transitionen fra det imperative programmeringsparadigme til funktionsprogrammeringsparadigmet. Hvor imperativ programmering er som en kageopskrift og kendes på brug af variable, for- og while-løkker, så lægger funktionsprogrammering vægt på funktioner og kendes på værdier og rekursion.

I denne periode kigger på særligt på 2 ny datastrukturer arrays og lists. De er begge datastrukturer til at holde lister af elementer af samme type, men arrays er variable og lists er værdier. Derfor ser man også sjældent arrays i programmer, der følger funktionsprogrammeringsparadigmet. I denne periode vil vi lægge vægt på programmering af arrays og lists via de medfølgende `List` og `Array` moduler.

Emnerne for denne arbejdsseddel er:

- at kunne oprette, gennemløbe og lave beregninger med lists vha. `List`-modulet og med `for-in`-løkker,
- at kunne oprette, gennemløbe og ændre arrays vha. `Array`- og `Array2D`-modulet,
- at kunne beskrive de kvalitative forskelle mellem lists og arrays.

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

Øveopgaver (in English)

5ø0 Explain the difference between the types `int -> (int -> int)` and `(int -> int) -> int`, and give an example of a function of each type.

5ø1 Write the types for the functions `List.filter` and `List.foldBack`.

5ø2 Write a function oneToN : n:int -> int list which returns the list of integers [1; 2; ...; n].

5ø3 Write a function printLstAlt: 'a list -> (), which uses for-in, to print every element of a given list to the screen. Ensure that your function works for lists of various types, e.g., int list and string list.

5ø4 Write a function printLst: 'a list -> (), which uses List.iter, an anymous function, and printfn "%A" to print every element of a given list to the screen. Ensure that your function works for lists of various types, e.g., int list and string list.

5ø5 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].

5ø6 Make a function even: int -> bool which returns true if the input is even and false otherwise. Use List.filter and even to make another function filterEven : int list -> int list, which returns all the even numbers of a given list.

5ø7 Write a function multiplicity: x:int -> xs:int list -> int, which counts the number of occurrences of the number x in the list xs using List.filter, an anonymous function, and the Length property.

5ø8 Write a function rev: 'a list -> 'a list, which uses List.fold, an anymous function, and the "::" operator to reverse the elements in a list. Ensure that your function works for lists of various types, e.g., int list and string list.

5ø9 Use List.map to make a function applylist : ('a -> 'b) list -> 'a -> 'b list, which applies a list of functions to the same element and returns a list of results. For example applylist [cos; sin; log; exp] 3.5 should return approximately [-0.94; -0.35; 1.25; 33.11].

5ø10 Use Array.init to make a function squares: n:int -> int [], such that the call squares n returns the array of the first n square numbers. For example, squares 5 should return the array [|1; 4; 9; 16; 25|].

5ø11 Write a function reverseArray : arr:'a [] -> 'a [] using Array.init and Array.length which returns an array with the elements in the opposite order of arr. For eksample, printfn "%A" (reverseArray [|1..5|]) should write [|5; 4; 3; 2; 1|] to the screen.

5ø12 Write the function reverseArrayD : arr:'a [] -> unit, which reverses the order of the values in arr using a while-loop to overwrite its elements. For example, the program

```
let aa = [|1..5|]
reverseArrayD aa
printfn "%A" aa
```

should output [|5; 4; 3; 2; 1|].

Afleveringsopgaver (in English)

5g0 A table can be represented as a non-empty list of equally long lists, for example, the list `[[1; 2; 3]; [4; 5; 6]]` represents the table:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

(a) Make a function `isTable : llst:'a list list -> bool`, which determines whether `llst` is a legal non-empty list, i.e., that

- there is at least one element, and
- all lists in the outer list has equal length.

(b) Make a function `firstColumn : llst:'a list list -> 'a list` which takes a list of lists and returns the list of first elements in the inner lists. For example, `firstColumn [[1; 2; 3]; [4; 5; 6]]` should return `[1; 4]`. If any of the lists are empty, then the function must return the empty list of integers `[] : int list`.

(c) Make a function `dropFirstColumn : llst:'a list list -> 'a list list` which takes a list of lists and returns the list of lists where the first element in each inner list is removed. For example, `dropFirstColumn [[1; 2; 3]; [4; 5; 6]]` should return `[[2; 3]; [5; 6]]`. Ensure that your function fails gracefully, if there is no first elements to be removed.

(d) Make a function `transposeLstLst : llst:'a list list -> 'a list list` which transposes a table implemented as a list of lists, that is, an element that previously was at `a.[i,j]` should afterwards be at `a.[j,i]`. For example, `transposeLstLst [[1; 2; 3]; [4; 5; 6]]` should return `[[1; 4]; [2; 5]; [3; 6]]`. Ensure that your function fails gracefully. Note that `transposeLstLst (transposeLstLst t) = t` when `t` is a table as list of lists. Hint: the functions `firstColumn` and `dropFirstColumn` may be useful.

(e) Make a whitebox test of the above functions.

5g1 Arrays are an alternative data structure for tables.

(a) Use `Array2D.init`, `Array2D.length1` and `Array2D.length2` to make the function `transposeArr : 'a [,] -> 'a [,]` which transposes the elements in input.

(b) Make a whitebox test of `transposeArr`.

(c) Comparing this implementation with Assignment 5g0d, what are the advantages and disadvantages of each of these implementations?

(d) For the application of tables, which of lists and arrays are better programmed using the imperative paradigm and using the functional paradigm and why?

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder 5g_<navn>.zip (f.eks. 5g_jon.zip)

Zip-filen 5g_<navn>.zip skal indeholde en src mappe og filen README.txt. I src skal der ligge følgende og kun følgende filer: 5g0.fsx og 5g1.fsx svarende til hver af delopgaverne. De skal kunne oversættes med fsharpc, og de oversatte filer skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden som minimum ved brug af <summary>, <param> og <returns> XML-tagsne. Udover selve koden skal besvarelser indtastes som kommentarer i de fsx-filer, de hører til. Filen README.txt skal ganske kort beskrive, hvordan koden oversættes og køres, og eventuelle Black-box, White-box og håndkøringsresultater når relevant.

God fornøjelse.