Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

September 16, 2022

0.1 Lists

0.1.1 Opgave(r)

0.1.1: Write a recursive function `oneToN : n:int -> int list` which uses the cons operator, `::`, and returns the list of integers `[1; 2; ...; n]`.

0.1.2: Skriv en funktion `multiplicity: x:int -> xs:int list -> int`, som tæller antallet af gange tallet `x` optræder i listen `xs`.

0.1.3: Write a function `split: xs:int list -> (xs1: int list) * (xs2: int list)` which separates the list `xs` into two and returns the result as a tuple where all the elements with even index is in the first element and the rest in the second. For example, `split [x0; x1; x2; x3; x4]` should return `([x0; x2; x4], [x1; x3])`.

0.1.4: Definer en funktion `reverseApply : x:'a -> f:('a -> 'b) -> 'b`, sådan at kaldet `reverseApply x f` returnerer resultatet af funktionsanvendelsen `f x`.

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

0.1.6: Brug `List.filter` til at lave en funktion `evens : lst:int list -> int list`, der returnerer de lige heltal i liste `lst`.

0.1.7: Brug `List.map` og `reverseApply` (fra Opgave 4) til at lave en funktion `applylist : lst:('a -> 'b) list -> x:'a -> 'b list`, der anvender en liste af funktioner `lst` på samme element `x` for at returnere en liste af resultater.

0.1.8: Write the types for the functions `List.filter` and `List.foldBack`.

0.1.9: En snedig programmør definerer en sorteringsfunktion med definitionen `ssort xs = Set.toList (Set.ofList xs)`. For eksempel giver `ssort [4; 3; 7; 2]` resultatet `[2; 3; 4; 7]`. Diskutér, om programmøren faktisk er så snedig, som han tror.

0.1.10: 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|]`.

0.1.11: 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.

0.1.12: 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|]`.

0.1.13: 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 14d, 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?

0.1.14: 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.

0.1.15: Brug funktionerne opremset i [Kapitel 11, Sporring] til at definere en funktion `concat : 'a list list -> 'a list`, der sammensætter en liste af lister til en enkelt liste. F.eks. skal `concat [[2]; [6; 4]; [1]]` give resultatet `[2; 6; 4; 1]`.

0.1.16: Brug funktionerne fra [Kapitel 11, Sporring] til at definere en funktion `gennemsnit : float list -> float option`, der finder gennemsnittet af en liste af kommatal, såfremt dette er veldefineret, og `None`, hvis ikke.