02157 Functional programming

Michael R. Hansen
DTU Compute
October 30, 2018

Mandatory assignment 3

This is the third of four mandatory assignments in 02157 Functional programming. It is a requirement for exam participation that 3 of the 4 mandatory assignments are approved. The mandatory assignments can be solved individually or in groups of 2 or 3 students.

Acceptance of a mandatory assignment from previous years does NOT apply this year.

- Your solution should be handed in **no later than Thursday, November 8, 2018**. Submissions handed in after the deadline will face an *administrative rejection.*
- You should solve the programming problem on the subsequent pages. A solution should have the form of a single F# file (*file*.`fsx` or *file*.`fs`). In your solution you are allowed to introduce helper functions; but you must also provide a declaration for each of the required functions, so that it has exactly the type and effect asked for.
- Each function declaration should be accompanied by a few illustrative test cases.
- Do not use imperative features, like assignments, arrays and so on in your programs. Failure to comply with that will result in an *administrative rejection of your submission.*
- To submit you should upload a single F# file (*file*.`fsx` or *file*.`fs`) to Inside under Assignment 3.

  The file should start with full names and study numbers for all members of the group. If the group members did not contribute equally to the solution, then the role of each student must be explicitly stated.
- Your F# solution must be a complete program that can be uploaded to F# Interactive without encountering compilation errors. Failure to comply with that will result in an *administrative rejection of your submission.*
- Be careful that you submit the right file. A submission of a wrong file will result in an *administrative rejection of your submission.*
- **DO NOT COPY solutions** from others and **DO NOT SHARE your solution** with others. Both cases are considered as fraud and will be reported.

Structured Documents

In this assignment we shall consider structured text documents, having paragraphs and (sub)sections, as modelled by the type declarations:

```
type Title    = string
type Document = Title * Element list
and  Element  = Par of string | Sec of Document;;
```

A *document* is a pair (t, es) consisting of a title t and a list of elements es, where an element can be a paragraph (constructor `Par`) characterized by a string or a (sub)section (constructor `Sec`) characterized by a document. An example is:

```
let s1   = ("Background", [Par "Bla"])
let s21  = ("Expressions", [Sec("Arithmetical Expressions", [Par "Bla"]);
                            Sec("Boolean Expressions", [Par "Bla"])])
let s222 = ("Switch statements", [Par "Bla"])
let s223 = ("Repeat statements", [Par "Bla"])
let s22  = ("Statements",[Sec("Basics", [Par "Bla"]) ; Sec s222; Sec s223])
let s23  = ("Programs", [Par "Bla"])
let s2   = ("The Programming Language", [Sec s21; Sec s22; Sec s23])
let s3   = ("Tasks", [Sec("Frontend", [Par "Bla"]);
                      Sec("Backend", [Par "Bla"])])
let doc  = ("Compiler project", [Par "Bla"; Sec s1; Sec s2; Sec s3]);;
```

where `doc` describes a document with title `"Compiler project"`. This document has three sections, where, for example, the section with title `"Statements"` has a subsection with the title `"Repeat statements"`, and so on.

Hint: Notice the mutual recursion in the declarations of the types `Document` and `Element`. You may consider using mutually recursive auxiliary functions in your solutions to the below questions.

1. Declare a function `noOfSecs` *d* that counts the number of sections (including subsections) in the document *d*. For example, `noOfSecs doc` is 13 (the number occurrences of constructor `Sec` in the value of `doc`).
2. Declare a function `sizeOfDoc` *d* that gives the number of characters in document *d*, that is, the sum of the lengths of all strings occurring in titles and paragraphs in *d*.
3. Declare a function `titlesInDoc` *d* that gives a list containing all the titles of sections and subsections occurring in document *d*. For example, the value of `titlesInDoc doc` should contain 13 strings including `"Backend"` and `"Background"`; but it should *not* contain `"Compiler Project"` as this is the title of the entire document and not a section title.

We shall use integer lists, called *prefixes*, to identify sections in documents in the way you are familiar with from text documents. The empty prefix, that is [], identifies the title of the entire document. The *table of contents* of a document is a list of pairs of prefixes and titles (of matching (sub)sections or of the entire document):

```
type Prefix = int list;;
type ToC    = (Prefix * Title) list
```

For example, the subsection with title `"Arithmetical Expressions"` is identified by the prefix [2; 1; 1] as it occurs in the first subsection of the first subsection of the second section of `doc`. The title for this subsection could appear as **2.1.1 Arithmetical Expressions** in a text document. The table of contents for `doc` is

```
[([], "Compiler project");
 ([1], "Background");
 ([2], "The Programming Language");
 ([2;1], "Expressions");
 ([2;1;1], "Arithmetical Expressions");
 ([2;1;2], "Boolean Expressions");
 ([2;2], "Statements");
 ([2;2;1], "Basics");
 ([2;2;2], "Switch statements");
 ([2;2;3], "Repeat statements");
 ([2;3], "Programs");
 ([3], "Tasks");
 ([3;1], "Frontend");
 ([3;2], "Backend")]
```

4. Declare a function `toc: Document` $\rightarrow$ `ToC` that generates the table of contents for a document.

   Hint: You may consider using mutually recursive auxiliary functions that may take prefixes and (in some cases) section counters as extra arguments.