Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 3 - gruppeopgave

Jon Sporring

14. september - 19. september.
Afleveringsfrist: lørdag d. 19. september kl. 22:00.

1 Organisere kode i funktioner og løkker

At programmere er kunsten at løse et problem vha. et program, dvs. at tænke i programmeringsstrukturer, og et godt program kendetegnes ved, at programmet er let at vedligeholde og udvide for programmøren selv og for andre. Til det benytter vi forskellige programmeringselementer, f.eks. betingelser og løkker til at styre om programdele skal køres 0, 1 eller flere gange, funktioner til at strukturere koden i let forståelige, genbrugbare og isolerede kodeenheder og kommentarer til at beskrive tanker bag kritiske programmeringselementer til sig selv og til andre.

Emnerne for denne arbejdsseddel er:

- Producere text output på skærmen og modtage input fra tastaturet,
- Organisere kode ved brug af funktioner,
- Kontrolere programflow med betingelser og løkker
- Dokumentere programmer ved hjælp af kommentarer i koden.

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)

3ø0 Enter the following program in a text file, compile, and execute the program:

Listing 1: Value bindings.

```
let a = 3
let b = 4
let x = 5
printfn "%A * %A + %A = %A" a x b (a * x + b)
```

Explain why the the parenthesis in the call to `printfn` is necessary. Add a line, which calculates the expression $ax+b$ and binds the result to the name `y`. Modify the call to `printfn`, such that it uses this new name. Is it still necessary to use parentheses?

3ø1 Listing 1 uses F#'s Lightweight syntax. Rewrite the program with the `y`-binding, such that it uses regular syntax.

3ø2 The following program,

Listing 2: Strenge.

```
let firstName = "Jon"
let lastName = "Sporring" in let name = firstName + " " +
   lastName;;
printfn "Hello %A!" name;;

```

is supposed to write "Hello Jon Sporring!" to the screen, but unfortunately, it contains at least one mistake. Correct the mistake(s). Rewrite the program into a one-line program without the use of semicolons. Consider how many ways this can be done, where you still use the bindings `firstName`, `lastName`, `name`, and the `printfn` function.

3ø3 Add the function

```
f : a:int -> b:int -> x:int -> int
```

to Listing 1 where `a`, `b`, and `x` are arguments to the expression $ax+b$, and modify the call to `printfn` such that it uses the function instead of the expression.

3ø4 Using the function developed in Assignment 3ø3, and print its value for $a = 3$, $b = 4$, and $x = 0\ldots5$ using:

(a) 6 `printfn`-statements,

(b) a `for`-loop and a single `printfn`-statement,

(c) a `while` løkke and a single `printfn`-statement,

Which version is simplest simplest to update, in case we later want to change the range of x?

3ø5 Consider the faculty-function,

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \ldots \cdot n \tag{1}$$

(a) Write a function

```
fac : n:int -> int
```

which uses a while-loop, a counter variable, and a local variable to calculate the faculty-function as (1).

(b) Write a program, which asks the user to enter the number `n` using the keyboard, and which writes the result of `fac n`.

(c) Make a new version,

```
fac64 : n:int -> int64
```

which uses `int64` instead of `int` to calculate the faculty-function. What are the larges values n, for which `fac` and `fac64` respectively can calculate the faculty-function for?

Afleveringsopgaver (in English)

3g0 Consider the following sum of integers,

$$\sum_{i=1}^{n} i. \tag{2}$$

This assignment has the following sub-assignments:

(a) Write a function

```
sum : n:int -> int
```

which uses the counter value, a local mutable value `s`, and a while-loop to compute the sum $1+2+\cdots+n$ as (2). If the function is called with any value smaller than 1, then it is to return the value 0.

(b) By induction one can show that

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, n \geq 0 \tag{3}$$

Make a function

```
simpleSum : n:int -> int
```

which uses (3) to calculate $1+2+\cdots+n$ and which includes a comment explaining how the expression implemented is related to the mentioned sum.

(c) Write a program, which asks the user for the number n, reads the number from the keyboard, and write the result of `sum n` and `simpleSum n` to the screen.

(d) Make a program, which write a table to the screen with 3 columns: `n`, `sum n` og `simpleSum n`. The table should have a row for each of $n = 1, 2, 3, .., 10$. Verify that the two functions calculate identical results.

(e) What is the largest value n that the two sum-functions can correctly calculate the value of? Can the functions be modified, such that they can correctly calculate the sum for larger values of n?

3g1 Consider multiplication tables of the form,

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	19	20
3	3	6	9	12	15	18	21	24	27	30
...										

where the elements of the top row and left column are multiplied and the result is written at their intersection.

In this assignment, you are to work with a function

```
mulTable : n:int -> string
```

which takes 1 argument and returns a string containing the first $1 \leq n \leq 10$ lines in the multiplication table including `<newline>` characters. The resulting string must be printable with a single`printf "%s"`statement. For example, the call `mulTable 3` must return.

Listing 3: An example of the output from `mulTable`.

```
printf "%s" (mulTable 3);;
         1    2    3    4    5    6    7    8    9   10
    1    1    2    3    4    5    6    7    8    9   10
    2    2    4    6    8   10   12   14   16   18   20
    3    3    6    9   12   15   18   21   24   27   30
```

All entries must be padded with spaces such that the rows and columns are right-aligned. Consider the following sub-assignments:

(a) Make

```
mulTable : n:int -> string
```

such that it has one and only one value binding to a string, which is the resulting string for $n = 10$, and use indexing to return the relevant tabel for $n \leq 10$. Test `mulTable n` for $n = 1, 2, 3, 10$. The function should return the empty string for values $n < 1$ and $n > 10$.

(b) Make

```
loopMulTable : n:int -> string
```

such that it uses a local string variable, which is built dynamically using 2 nested `for`-loops and the `sprintf`-function. Test `loopMulTable n` for $n = 1, 2, 3, 10$.

(c) Make a program, which uses the comparison operator for strings, “=”, and write a table to the screen with 2 columns: `n`, and the result of comparing the output of `mulTable n` with `loopMulTable n` as `true` or `false`, depending on whether the output is identical or not.

(d) Use `printf "%s"`and `printf "%A"`to print the result of `mulTable`, and explain the difference.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil

Zip-filen skal indeholde en `src` mappe og filen `README.txt`. Mappen skal indeholde fsharp koden, der skal være en fsharp tekstfil per fsharp-opgave, og de skal navngives `3g0.fsx` osv. De skal kunne oversættes med fsharpc, og de oversatte filer skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden, og udover selve programteksten skal besvarelserne indtastes som kommentarer i de fsx-filer, de hører til. Filen `README.txt` skal ganske kort beskrive, hvordan koden køres.

God fornøjelse.