Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 4 - individuel opgave

Jon Sporring

21. september - 26. september.
Afleveringsfrist: lørdag d. 26. september kl. 22:00.

Organisere kode i moduler og gennemføre en afprøvning

Der er (næsten) altid fejl i kode, og retter man en, er der en væsentlig sandsynlighed for at man tilføjer en anden. Fejlfindingsprocessen er derfor en essentiel del af programmering, og i denne periode kigger vi på 3 forskellige metoder: Black-box, hvor man fokuserer på opfyldelse af de ydre krav til en funktion og et program, og som typisk er stillet af en ekstern opgavestiller; White-box, hvor man fokuserer kodens interne opbygning; og håndkøring, hvor man simulerer computerens udførsel af et program. Chancer for fejlfinding og genbrug kan øges, hvis man organiserer sin kode godt. I sidste uge kiggede vi på, hvordan løkker og funktioner kan bruges til kodeorganisation, i denne uge vil vi kigge på, hvordan man samler funktioner i biblioteker (moduler) efter tema, og hvordan man i den forbindelse kan bruge signaturfiler sammen med kommentarer til at skabe et klart overblik over, hvad funktioner i moduler kan uden at man lader sig forvirre eller opsluge af deltajer i funktionernes implementation.

Emnerne for denne arbejdsseddel er:

- Kunne strukturere kode med moduler,
- Kunne tilføje veldokumenterede signaturfiler til moduler,
- Kunne gennemføre en white- og black-box afprøvning og håndkøring.

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)

A color is often represented as a triple (red, green, blue), where each entry is called a color-channel, and each channel is typically an integer between and including 0 and 255:

$$c = (r, g, b). \tag{1}$$

Colors can be added, by adding their channels,

$$\begin{align} c_1 + c_2 &= \big(\operatorname{trunc}(r_1 + r_2), \operatorname{trunc}(g_1 + g_2), \operatorname{trunc}(b_1 + b_2)\big), \tag{2}\\ c_i &= (r_i, g_i, b_i), \tag{3}\\ \operatorname{trunc}(v) &= \begin{cases} 0, & v < 0 \\ 255, & v > 255 \\ v, & \text{ellers} \end{cases} \tag{4} \end{align}$$

and colors can be scaled by a factor by multiplying each channel with that same factor,

$$ac = \big(\operatorname{trunc}(ar), \operatorname{trunc}(ag), \operatorname{trunc}(ab)\big). \tag{5}$$

Colors where the channels have identical values, $v = r = g = b$, are grays, and colors are converted to grays as the average,

$$v = \operatorname{gray}(c) = \frac{r + g + b}{3}. \tag{6}$$

4ø0 Write a signature file for a module which contains the functions `trunc`, `add`, `scale`, and `gray` from the mathematical definitions above and by use of tuples where possible.

4ø1 Write an implementation of the signatur file from Assignment 4ø0 and compile both files into a library (`dll`-fil).

4ø2 Write two programs: One which uses the library developed in Assignment 4ø0 and 4ø1 using `fsharpi` and one which uses `fsharpc`.

4ø3 Make a Black-box test of your library from Assignment 4ø1 and with the use of `fsharpc`.

4ø4 Make a White-box test of your library from Assignment 4ø1 and with the use of `fsharpc`.

4ø5 Consider the library from Assignment 4ø1. Assuming that your module is called `Color`, consider the following application

Listing 1: Application of a Color library.

```
let red = (255,0,0)
let green = (0,255,0)
let avg = Color.add red green
let factor = 1.25
let bright = Color.scale factor avg
printfn "Bright gray is: %A" bright
```

If your functions `add` and `scale` have a different interface, then adjust accordingly. Perform a tracing by hand of the above code including the implementation of your library. Run the (adjusted) code with `fsharpc`. Did you discover any errors? Do you get the same output?

Afleveringsopgaver (in English)

This assignment is about 2-dimensional vectors. A 2-dimensional vector (henceforth just called a vector) is a geometrical object consisting of a length and a direction. Typically, a vector is represented as a pair of numbers, $\vec{v} = (x, y)$, where its length and direction are found as,

$$\operatorname{len}(\vec{v}) = \sqrt{x^2 + y^2} \tag{7}$$

$$\operatorname{ang}(\vec{v}) = \operatorname{atan2}(y, x) \tag{8}$$

Vectors are often drawn as arrows with a head and a tail. In the Cartesian coordinate system, if the tail is placed at $(0, 0)$, then the head will be at (x, y). There exists a number of standard operations for vectors:

$$\vec{v}_1 = (x_1, y_1) \tag{9}$$

$$\vec{v}_2 = (x_2, y_2) \tag{10}$$

$$a\vec{v}_1 = (ax_1, ay_1) \tag{11}$$

$$\vec{v}_1 + \vec{v}_2 = (x_1 + x_2, y_1 + y_2) \tag{12}$$

$$\vec{v}_1 \cdot \vec{v}_2 = x_1 x_2 + y_1 y_2 \tag{13}$$

Addition can also be drawn, as shown in Figure 1.

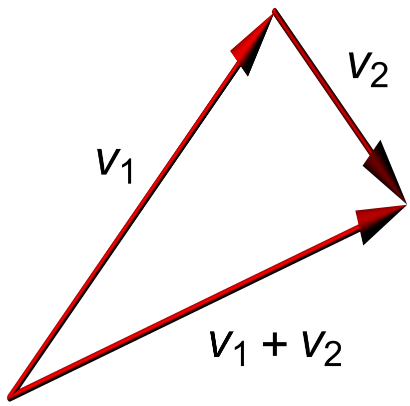

Figure 1: An illustation of vector addition.

4i0 Consider the signature file given in Listing 2 which contains some of the standard operations for vectors.

Listing 2 vec2dsmall.fsi:
A signature file for vector operations.

```
/// A 2 dimensional vector library.
/// Vectors are represented as pairs of floats
module vec2d
/// The length of a vector
val len : float * float -> float
/// The angle of a vector
val ang : float * float -> float
/// Addition of two vectors
val add : float * float -> float * float -> float * float
```

Solve the following sub-tasks:

a) Extend the signature file with documentation using the documentation standard.

b) Write a library vec2dsmall.fs implementing the signatures.

c) Compile the signature and the implementation into vec2dsmall.dll demonstrating that there are no syntax errors.

4i1 Write a Black-box test of the library.

4i2 Write a White-box test of the library.

4i3 Consider the following application

Listing 3: Application of a Color library.

```
let v = (1.3, -2.5)
printfn "Vector %A: (%f, %f)" v (vec2d.len v) (vec2d.ang v)
let w = (-0.1, 0.5)
printfn "Vector %A: (%f, %f)" w (vec2d.len w) (vec2d.ang w)
let s = vec2d.add v w
printfn "Vector %A: (%f, %f)" s (vec2d.len s) (vec2d.ang s)

```

First run the code with fsharpc. Then perform a tracing by hand of the above code including the implementation of your library. Did you discover any errors? Do you get the same output?

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder 4i_<navn>.zip (f.eks. 4i_jon.zip)

Zip-filen 4i_<navn>.zip skal indeholde en src mappe og filen README.txt. I src skal der ligge følgende og kun følgende filer: vec2d.fsi, vec2d.fs, 4i1.fsx, 4i2.fsx, 4i3.fsi 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.

God fornøjelse.