Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 2 - gruppeopgave

Jon Sporring

19. - 24. september.
Afleveringsfrist: lørdag d. 24. september kl. 22:00.

Med denne arbejdsseddel skifter vi langsomt fokus 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 vi 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 strukturere kode vha. funktioner,
- problemløsning vha. design med 8-trinsmodellen
- at kunne håndkøre simple programmer.

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)

2ø0 Consider the factorial-function,

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \ldots \cdot n \qquad (1)$$

(a) Write a function

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

which uses recursion to calculate the factorial-function as (1). The function should not use any variables.

(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 factorial-function. What are the largest values n, for which `fac` and `fac64` respectively can calculate the factorial-function for?

2ø1 Write a function `pow2 : n:int -> int` that calculates the value 2^n, assuming $n \geq 0$, using recursion. The function should return the value 1 if it is passed a negative value.

2ø2 Write a function `powN : x:int -> n:int -> int` that calculates the value x^n, assuming $n \geq 0$, using recursion. The function should return the value 1 if it is passed a negative value as its second argument.

2ø3 We can represent 2-dimensional vectors as tuples, as seen in the below code snippet:

```
let myVector = (1,2)
let zeroVector = (0,0)
```

(a) Write a function:

```
createIntVec : x:int -> y:int -> int * int
```

that given two integers x and y returns a vector represented as a tuple, `(x, y)`.

(b) Write a function:

```
createFloatVec : x:float -> y:float -> float * float
```

that given two floats x and y returns a vector of floats.

(c) Write a function:

```
createFloatVecFromInts : x:int -> y:int -> float * float
```

that given two integers x and y returns a vector of floats.

(d) Write a function:

```
intToFloatVec : int * int -> float * float
```

that given a vector of two integers, returns a vector of floats.

(e) Write a function:

```
floatToIntVec : float * float -> int * int
```

that given a vector of two floats, returns a vector of ints.

(f) Write a function:

```
addIntVecs : int * int -> int * int -> int * int
```

that given two vectors of ints, adds them with vector addition and returns the resulting vector.

(g) Write a function:

```
subIntVecs : int * int -> int * int -> int * int
```

that given two vectors of ints, subtracts them with vector subtraction and returns the resulting vector.

(h) Write a function:

```
scaleIntVec : int * int -> scalar:int -> int * int
```

that given a vector of ints and a scalar, scales the vector and returns the resulting vector.

Afleveringsopgaver (in English)

2g0 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 pattern-matching and recursion to compute the sum $1+2+\cdots+n$ also written in (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 writes a table to the screen with 3 columns: `n`, `sum n` and `simpleSum n`. The table should have a row for each of $n = 1, 2, 3, .., 10$, and each field must be 4 characters wide. Verify programmatically 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?

2g1 Building upon the exercises concerning vectors and their representation as tuples, we will construct a small library of functions for 2-dimensional vector operations, with vectors represented as tuples of floats.

(a) Write a function:

```
scaleFloatVec : float * float -> scalar:float -> float * float
```

that given a vector of floats and a scalar, scales the vector and returns the resulting vector.

(b) Write a function:

```
addFloatVecs : float * float -> float * float -> float * float
```

that given two vectors of floats, adds them and returns the resulting vector.

(c) Write a function:

```
subFloatVecs : float * float -> float * float -> float * float
```

that given two vectors of floats, subtracts them with vector subtraction and returns the resulting vector.

(d) Write a function:

```
lengthFloatVec : float * float -> float
```

that given a vectors of floats, computes the length of the vector and returns the result as a float.

(e) Write a function:

```
dotFloatVecs : float * float -> float * float -> float
```

that given two vectors of floats, computes the dot product and returns the result as a float.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `2g_<navn>.zip` (f.eks. `2g_jon.zip`)

Zip-filen `2g_<navn>.zip` skal indeholde en `src` mappe, filen `README.txt` *og filen "group.txt", der indeholder jeres kuid'er, ét per linje*. I `src` skal der ligge følgende og kun følgende filer: `2g0.fsx` og `2g1.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.