Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 6 - individuel opgave

Jon Sporring

5. oktober - 10. oktober.
Afleveringsfrist: lørdag d. 10. oktober kl. 22:00.

Løkker i funktionsprogrammering foretages ofte ved rekursive kald, som er et kald af en funktion til funktionen selv, som f.eks. fibonacci talrækken, $f(1) = f(2) = 1, f(i) \stackrel{i \geq 2}{=} f(i-1) + f(i-2)$. Rekursion er et kraftigt værktøj til løsning af en række problemer bla. ved behandling af lister, og kan kædes direkte sammen med induktionsbeviser til at bevise kodes korrekthed. Det er også en tankegang, som det tager tid at tilegne sig, og derfor bruger vi denne periode på udelukkende at arbejde med rekursive funktioner.

Emnerne for denne arbejdsseddel er:

- Skrive en rekursiv funktion og forklare forskellen på alm. og halerekurssion,
- Gemmenløbe en liste imperativt og rekursivt med patterns, og forklare fordele og ulemper ved de 2 tilgange
- Løse et problem vha. rekursion
- Skrive funktioner, som bruger pattern matching til behandling af simple typer og lister

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)

6ø0 Write a function `fac : n:int -> int` that calcualtes the faculty function $n! = \prod_{i=1}^{n} i$ using recursion.

6ø1 Write a function, `sumRec : n:int -> int` that calculates the sum $\sum_{i=1}^{n} i$ using recursion. Extend your table from Assignment 3g0 with a column for `sumRec` and compare this function with `sum` and `simpleSum`.

6ø2 Write a function `sum : int list -> int` that takes a list of integers and returns their sum. The function must traverse the list using recursion and pattern matching.

6ø3 The greatest common divisor (gcd) between two integers t and n is the largest integer c that divides both t and n with 0 remainder. Euclid's algorithm[1] finds gcd using the recursion:

$$\gcd(t,0) = t, \tag{1}$$
$$\gcd(t,n) = \gcd(n, t \mathbin{\%} n), \tag{2}$$

where % is the remainder operatorer (as in F#).

(a) Implement Euclid's algorithm by recursion

```
gcd : t:int -> n:int -> int
```

(b) Make a white- and blackbox test of your implemenation.

(c) Make a tracing by hand of the algorithm for `gcd 8 2` and `gcd 2 8`.

6ø4 Make your own implementation of `List.fold` and `List.foldback` using recursion.

6ø5 Use `List.fold` and `List.foldback` and your own implementations from Assignment 6ø4 to calculate the sum of the elements in `[0 .. n]`, where `n` is a large number.

Afleveringsopgaver (in English)

In this assignment, you will work with continued fractions[2]. Continued fractions are lists of integers that represents real numbers. The list is finite for rational numbers and infinte for irrational numbers.

Continued fractions to decimal numbers A continued fraction is written as $x = [q_0; q_1, q_2, \ldots]$ and the corresponding decimal number is found by the following recursive algorithm:

$$x = q_0 + \frac{1}{q_1 + \frac{1}{q_2 + \ldots}}. \tag{3}$$

The series of fraction continues as long as there are elements in the continued fraction.

For example, $[3;4,12,4] = 3.245$, since:

$$x = 3 + \frac{1}{4 + \frac{1}{12 + \frac{1}{4}}} \tag{4}$$
$$= 3 + \frac{1}{4 + \frac{1}{12.25}} \tag{5}$$
$$= 3 + \frac{1}{4.081632653} \tag{6}$$
$$= 3.245. \tag{7}$$

[1] https://en.wikipedia.org/wiki/Greatest_common_divisor

[2] https://en.wikipedia.org/wiki/Continued_fraction

Decimal numbers to continued fractions For a given number x on decimal form, its continued fraction $[q_0; q_1, q_2, \ldots]$ can be found using the following algorithm:

Let $x_0 = x$ and $i \geq 0$, and calculate

$$q_i = \lfloor x_i \rfloor \tag{8}$$
$$r_i = x_i - q_i \tag{9}$$
$$x_{i+1} = 1/r_i \tag{10}$$
$$\tag{11}$$

recursively until $r_i = 0$. The continued fraction is then the sequences of q_i.

For example, if $x = 3.245$ then

i	x_i	$q_i = \lfloor x_i \rfloor$	$r_i = x_i - q_i$	$x_{i+1} = 1/r_i$
0	3.245	3	0.245	4.081632653...
1	4.081632653...	4	0.081632653	12.25
2	12.25	12	0.25	4
3	4	4	0	-

and hence, the continued fraction is in the third column as $3.245 = [3; 4, 12, 4]$.

6i0 Write a recursive function

```
cfrac2float : lst:int list -> float
```

that takes a list of integers as a continued fraction and returns the corresponding real number.

6i1 Write a function

```
float2cfrac : x:float -> int list
```

that takes a real number and calculates its continued fraction.

6i2 Collect the above functions in a library as the interface file `continuedFraction.fsi` and implementation file `continuedFraction.fs`. Make a white- and blackbox test of these functions as the application `continuedFractionTest.fsx`.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `6i_<navn>.zip` (f.eks. `6i_jon.zip`)

Zip-filen `6i_<navn>.zip` skal indeholde en og kun en mappe `6i_<navn>`. I den mappe skal der ligge en `src` mappe og filen `README.txt`. I `src` skal der ligge følgende og kun følgende filer:

continuedFraction.fsi, continuedFraction.fs og 6i4.fsx svarende til de relevante delopgaver. 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.