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

Martin Elsman og 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, $f(1) = f(2) = 1, f(i) \stackrel{i>2}{=} f(i-1) + f(i-2)$. Rekursion er et kraftigt værktøj til løsning af en række problemer, blandt andet ved behandling af lister, og kan kædes direkte sammen med induktionsbeviser til at bevise korrekthed af kode. 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 almindelig rekursion og halerekursion
- Gemmenløbe en liste imperativt og rekursivt ved brug af pattern matching, og forklare fordele og ulemper ved de to tilgange
- Løse et problem ved hjælp af rekursion
- Skrive funktioner, som bruger pattern matching til behandling af lister og simple værdier

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:uint -> uint` that calculates the faculty function $n! = \prod_{i=1}^{n} i$ using recursion.

6ø1 Write a function `pow2 : n:int -> int` that calculates the value 2^n, assuming $n >= 0$, using recursion. The function should return the value 1 if it is passed a negative value.

6ø2 Write a function `powN : N:int -> n:int -> int` that calculates the value N^n, assuming $n >= 0$, using recursion. The function should return the value 1 if it is passed a negative value as its second argument.

6ø3 Write a function `sumRec : n:uint -> uint` that calculates the sum $\sum_{i=0}^{n} i$ using recursion.

6ø4 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.

6ø5 Write a function `length : 'a list -> int` that calculates the length of the argument list using recursion. The function should make use of pattern matching on lists.

6ø6 Write a tail-recursive function `lengthAcc : acc:int -> xs:'a list -> int` that calculates the sum of `acc` plus the length of `xs` using tail-recursion. For instance, a call `lengthAcc 0 xs` should return the length of `xs`, and a call `lengthAcc n []` should return the value `n`.

6ø7 The greatest common divisor (gcd) between two integers t and n is the largest integer c that divides both t and n with 0 as remainder. Euclid's algorithm[1] finds the greatest common divisor, using recursion, as follows:

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

Here % is the remainder operator (as in F#).

(a) Implement Euclid's algorithm by writing a recurive function

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

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

(c) Write down a tracing-by-hand derivation for the calls `gcd 8 2` and `gcd 2 8`.

6ø8 Write a function `lastFloat : float list -> float` that, using recursion, returns the last element of the argument list if the list is non-empty and returns the float value `NaN` if the argument list is empty.

For example, the call `lastFloat [2.1;4.2]` should return the float value `4.2` and the call `lastFloat []` should return the value `NaN`.

6ø9 Write a function `map : ('a -> 'b) -> 'a list -> 'b list` that takes a function and a list as arguments and returns a new list of the same length as the argument list with elements obtained by applying the supplied function on each of the elements of the argument list. The function should make use of recursion and pattern matching on lists.

As an example, a call `map (fun x -> x+2) [2;3]` should return the list `[4;5]`.

6ø10 Make your own implementation of `List.fold` and `List.foldback` using recursion.

Afleveringsopgaver (in English)

In this assignment, you will work with simple continued fractions[2], henceforth just called continued fractions. Continued fractions are lists of integers which represent real numbers. The list is finite for rational numbers and infinite for irrational numbers.

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

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

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 fractions continues as long as there are elements in the continued fraction.

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

$$\begin{aligned} x &= 3 + \frac{1}{4 + \frac{1}{12 + \frac{1}{4}}} && (4)\\ &= 3 + \frac{1}{4 + \frac{1}{12.25}} && (5)\\ &= 3 + \frac{1}{4.081632653} && (6)\\ &= 3.245. && (7) \end{aligned}$$

Note that all but the first digit must be larger than 0, e.g., $[1;0]$ is an illigal number, and that every rational number has exactly 2 representations $[q_0; q_1, \ldots, q_n] = [q_0; q_1, \ldots, (q_n - 1), 1]$ where the first is called the canoncial representation. E.g., $[2;3] = [2;2,1]$, since

$$2 + \frac{1}{3} = 2 + \frac{1}{2 + \frac{1}{1}}. \tag{8}$$

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

$$\begin{aligned} q_i &= \lfloor x_i \rfloor && (9)\\ r_i &= x_i - q_i && (10)\\ x_{i+1} &= 1/r_i && (11)\\ & && (12) \end{aligned}$$

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. Recall that floating-point numbers are inaccurate, so you should check that r_i is reasonably close to 0 instead of comparing it for equality to 0.0. For example, `abs ri < 1e-10`.

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 som hedder `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 `continuedFractionTest.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. Filen README.txt skal ganske kort beskrive, hvordan koden oversættes og køres.

God fornøjelse.