Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 6 - gruppeopgave

Jon Sporring

21. oktober - 5. november.
Afleveringsfrist: lørdag d. 5. november kl. 22:00.

På denne periode skal vi arbejde med en lidt større programmeringsopgave og kigge på højereordens funktioner og abstrakte datatyper.

Denne arbejdsseddels læringsmål er:

- Kunne definere funktioner som tager andre funktioner som argument og/eller giver funktioner som returværdi
- Kunne definere funktioner ved currying
- Kunne arbejde med de abstrakte datatyper: træer, mængder, og maps.

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)

I det følgene skal I arbejde med polynomier. Et polynomium af grad n skrives som

$$f(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n = \sum_{i=0}^{n} a_i x^i.$$

6ø0 Skriv en funktion `poly: float list -> float -> float`, der tager som argumenter (1) en liste `a` af koefficienter med `a.[i]` $= a_i$ og (2) en x-værdi for derefter at returnere polynomiets værdi. Afprøv funktionen ved at lave tabeller for et lille antal polynomier af forskellig grad med forskellige koefficienter og forskellige værdier for x, og validér den beregnede værdi.

6ø1 Definer en funktion `line : float -> float -> float -> float` ved brug af `poly`, således at `line a0 a1 x` beregner værdien for et 1. grads polynomium hvor `a0` $= a_0$, `a1` $= a_1$ og `x` $= x$. Afprøv funktionen ved at tabellere værdier for `line` med det samme sæt af koefficienter $a_0 \neq 0$ og $a_1 \neq 0$ og et passende antal værdier for x.

6ø2 Benyt Currying af `line` til at lave en funktion `theLine : float -> float`, hvor parametrene `a0` og `a1` er sat til det samme som brugt i Opgave 6ø1. Afprøv `theLine` tilsvarende som `line` afprøves i Opgave 6ø1.

De følgende opgaver omhandler integration. Integralet af næsten alle integrable funktioner kan approximeres som

$$\int_a^b f(x)\,dx \simeq \sum_{i=0}^{n-1} f(x_i)\Delta x,$$

hvor $x_i = a + i\Delta x$ og $\Delta x = \frac{b-a}{n}$.

6ø3 Skriv en funktion `integrate : n:int -> a:float -> b:float -> (f : float -> float) -> float`, hvis argumenter `n`, `a`, `b`, er som i ligningerne, og `f` er en integrabel 1 dimensionel funktion. Afprøv `integrate` på `theLine` fra Opgave 6ø2 og på `cos` med $a = 0$ og $b = \pi$. Udregn integralerne analytisk og sammenlign med resultatet af `integrate`.

6ø4 Funktionen `integrate` er en approximation, og præcisionen afhænger af n. Undersøg afhængigheden ved at udregne fejlen, dvs. forskellen mellem det analytiske resultat og approximationen for værdier af n. Dertil skal du lave to funktioner `integrateLine : n:int -> float` og `integrateCos : n:int -> float` vha. `integrate`, `theLine` og `cos`, hvor værdierne for a og b og f er fastlåste. Afprøv disse funktioner for $n = 1, 10, 100, 1000$. Overvej om der er en tendens i fejlen, og hvad den kan skyldes.

The following exercises are about expanding and using the following recursive sumtype, which can be used for modelling expression terms:

```
type expr = Const of int | Add of expr * expr | Mul of expr * expr
```

6ø5 Implement a recursive function `eval : expr -> int` that takes an expression value as argument and returns the integer resulting from evaluating the expression term. The expression `eval (Add(Const 3,Mul(Const 2,Const 4)))` should return the integer value `11`.

6ø6 Extend the type `expr` with a case for subtraction, extend the evaluator with a proper `match`-case for subtraction, and evaluate that your implementation works in practice.

6ø7 Extend the type `expr` with a case for division and refine the evaluator function `eval` to have type `expr->(int,string)result`. Evaluate that your implementation will propagate “Divide by zero” errors to the toplevel.

In the following exercises, we shall investigate the following recursive type definition for trees:

```
type 'a tree = Leaf of 'a | Tree of 'a tree * 'a tree
```

The tree type is generic in the type of information that can be installed in `Leaf` nodes.

6ø8 Write a function `leafs : 'a tree -> int` that returns the number of leaf nodes appearing in a tree. Evaluate that your function works as expected.

6ø9 Write a function

```
    find : provide: ('a -> bool) -> t: 'a tree -> 'a option
```

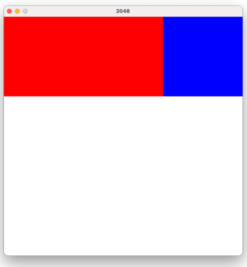
that, using a preorder traversal, returns the first value that satisfies the provided predicate. If no such value appears in the tree, the function should return the value `None`. Evaluate that your function works as expected.

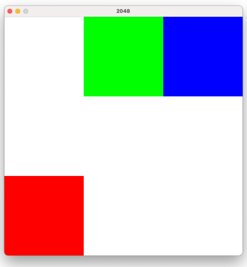
6ø10 Write a function `sum : int tree -> int` that returns the sum of the integer values appearing in the leafs of the tree. Evaluate that your function works as expected.

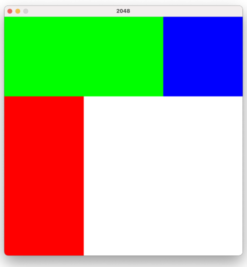
Afleveringsopgaver (in English)

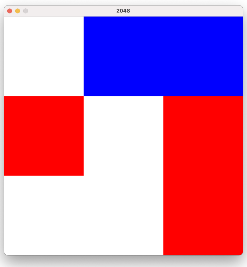
2048 is a popular solitaire board game available, e.g., online on `https://2048.io/`. In this exercise, you are to implement a version of this game in F# and using `Canvas`. The rules, you are to implement are as follows:

1. The board is a square board with 3×3 field.
2. The pieces are colored squares: red, green, blue, yellow, black corresponding to the values 2, 4, 8, 16, 32.
3. There can at most be one piece per field on the board.
4. The initial conditions is a red and a blue piece placed on the board.
5. The game can be tilted left, right, up, down by pressing the corresponding arrow keys.
6. When the game is tilted, then all the pieces are to be moved to the corresponding side of the board.
7. If two pieces of the same color touch after being tilted, then they are replaced by a single piece of double the value. For example, the board in Figure 1(a) is tilted to the right, and the two red pieces are replaced with green piece.
8. Identical pieces are combined in the order of the direction, they are tilted, e.g., if the board is tilted left, then identical pieces are identified from the left to right. Matching colors that touch in the direction of the tilt after a combination of pieces are not combined. E.g., the board in Figure 1(c) is tilted to the right combining the two green to a blue, but the resulting two blues are not combined.


(a)


(b)


(c)


(d)

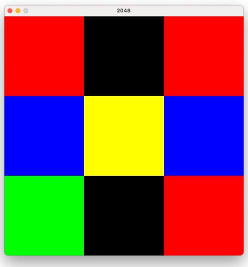

(e)

Figure 1: Some examples

9. Two black pieces are combined into one black piece.
10. After each turn, a new red piece is to be placed randomly on an available field on the board.
11. The game ends when there are no possible moves and no empty locations for a new piece to spawn.

In your solution, you are to represented a board with its pieces as a list of pieces, where each piece has a color and a position. This is captured by the following type abbreviations:

```
type pos = int*int // A 2-dimensional vector in board-coordinats (not
   pixels)
type value = Red | Green | Blue | Yellow | Black // piece values
type piece = value*pos //
type state = piece list // the board is a set of randomly organized
   pieces
```

6g0 Make a library consisting of a signature and an implementation file. The library must contain the following functions

```
// convert a 2048-value v to a canvas color
fromValue: v: value -> Canvas.color
// give the 2048-value which is the next in order from c
nextColor: c: value -> value
// return the list of pieces on a row r on board s
filter: r: int -> s: state -> state
// tilt all pieces on the board s to the left
shiftLeft: s: state -> state
```

```
// flip the board s such that all pieces position change as
   (i,j) -> (N-1-i,j)
fliplr: s: state -> state
// transpose the pieces on the board s such all piece positiosn
   change as (i,j) -> (j,i)
transpose: s: state -> state
// find the list of empty positions on the board s
empty: s: state -> pos list
// randomly place a new piece of color c on an empty position on
   the board s
addRandom: c: value -> s: state -> state option
```

With these functions and Canvas it is possible to program the game in a few lines. Add the following to your library:

(a) Write a canvas draw function

    draw: w: int -> h: int -> s: state -> canvas

  which makes a new canvas and draws the board in s.

(b) Write a canvas react function

    react: s: state -> k: key -> state option

  which titles the board base according to the arrow-key, the user presses. Note that tilt left is given by the `shiftLeft` function. Tilt right can be accomplished by `fliplr >> shiftLeft >> fliplr`, and tilt up and down can likewise be accomplished with the additional use of `transpose`.

Finally, make an application program, which calls `runApp "2048" 600 600 draw react board`.

All above mentioned functions are to be documented using the XML-standard, and simple test examples are to be made for each function showing that it likely works.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `6g.zip`
- en opgavebesvarelse i pdf-format.

Zip-filen skal indeholde:

- filen `README.txt` som er en textfil med jeres navne og dato arbejdet.
- en `src` mappe med følgende og kun følgende filer:

  6g0.fsi, 6g0.fs, 6g0.fsx

Funktionerne skal være dokumenteret med ifølge dokumentationsstandarden ved brug af `<summary>`, `<param>` og `<returns>` XML tagsne.

- pdf-dokumentet skal være lavet med LaTeX, benytte `rapport.tex` skabelonen, skal dokumentere jes løsning og indeholde figurer, der viser outputgrafik fra canvas for opgaverne.

God fornøjelse.