Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 5 - individuel opgave

Jon Sporring

7. oktober - 15. oktober.
Afleveringsfrist: lørdag d. 15. oktober kl. 22:00.

I denne periode skal vi arbejde med abstrakte datatype, biblioteker og applikationer. En abstrakt datatype er et koncept som f.eks. lister, som kan beskrives vha. dets interface med signaturfiler og implementeres som et bibliotek i implementationsfiler. Biblioteker i F# kaldes også for moduler. Denne arbejdsseddels læringsmål er:

- at lave et bibliotek og en tilhørende applikation,
- at kunne oversætte biblioteksfiler og applikationsfiler både med `dotnet fsi` og `run`,
- at kunne arbejde med generiske moduler,

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)

5ø0 In an earlier assignment, you implemented a small set of functions for vector operations in F#:

(a) addition of vectors

```
add: vec -> vec -> vec
```

(b) multiplication of a vector and a constant

```
mul: vec -> float -> vec
```

(c) rotation of a vector

```
rot: vec -> float -> vec
```

and wrote a small program to test these functions. Convert your earlier programs into a library called `Vec`, consisting of a signature file, an implementation file, and an application file. Create a `.fsproj` project file, and run the code using first `dotnet fsi` and then `dotnet run`.

Afleveringsopgaver (in English)

In the following, you are to work with the abstract datatype known as a *queue*. A queue consists of a sequence that supports the following operations: checking whether the sequence is empty; removing an element from the front ("left"); adding an element at the end ("right"). Queues appear often in real life: Standing in line at a shop counter, orders await in a queue for their turn to be shipped in an online shop, students waiting to be examined at oral examinations. *Purely functional queues* are a data type defined by their element type, the following set of operations, and the properties they must satisfy.

```
// types
type element // type of elements in the queue
type queue // type of queues with such elements
// values and functions
// the empty queue
val emptyQueue: queue
// add an element at the end of a queue
val enqueue: element -> queue -> queue
// check if the queue is empty
val isEmpty: queue -> bool
// remove the element at the front of the queue
// precondition: isEmpty(q) == false
val dequeue: queue -> element * queue
```

These queues are called *(purely) functional* because the enqueue and dequeue operations return a *new* queue they are called, without destroying the old queue. In particular, it is possible to add an element e_1 to a queue q_0 of length 15 resulting in a queue q_1 of length 16, then to add another element e_2 to q_0 resulting in a queue q_2 of length 16, but different from q_1 in the last element, whereupon we have 3 separate queues, each of which we can use in future operations: q_0, q_1 and q_2.[1]

In this exercise, you are to work with functional queues in F#. We'll leave off the "functional" below.

5i0 In the following, you are to implement your own queue module using lists in F# to represent the (sequence of elements in) a queue.

(a) Given the description of the abstract datatype Queue above, write a signature file for the functional queues. The module is to be called `queue`.

(b) Write an implementation file, implementing the signature file above using lists in F# where the elements are F# integers.

(c) Write an application file, which as minimum consists of the following series of tests of your queue module:

```
let q0 = emptyQueue
printfn "%A, %A, %A" q0 (isEmpty q0) (dequeue q0)
let q1 = create () |> enqueue 1 |> enqueue 2 |> enqueue 3
printfn "%A, %A" q1 (isEmpty q1)
let (e,q2) = dequeue q1
printfn "%A, %A" e q2
```

[1] There are also *ephemeral* (also called *imperative*) queues, where enqueue and replace the original queue with the new queue such that there is always just one "current" queue that changes over time. Ephemeral queues have more limited functionality and are easier to implement efficiently using imperative data structures, which we will encounter later in the course.

5i1 A problem with the queue specification above is that there is a precondition on the dequeue operation: A programmer must always ensure that the argument to dequeue is nonempty before calling dequeue. In other words, even though the F# type system does not flag it as an error, it *is* an error (by the programmer) to call dequeue with the empty queue.

Change the queue specification such that `dequeue` returns an `(element option) * queue` value and remove the precondition. Write an application demonstrating that your implementation works; that is, its operations perform queuing and dequeuing, and, additionally `dequeue(emptyQueue)` returns `None`.

5i2 Convert the integer queue into a generic queue, and demonstrate in your application, that you can build queues of different types with the same generic library.

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `5i.zip`
- en opgavebesvarelse i pdf-format.

Zip-filen skal indeholde:

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

  5i0.fsi, 5i0.fs 5i0.fsx,
  5i1.fsi, 5i1.fs 5i1.fsx,
  5i2.fsi, 5i2.fs 5i2.fsx,

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

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

God fornøjelse.