Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 8 - gruppeopgave

Jon Sporring

25. november - 3. december.
Afleveringsfrist: lørdag d. 3. december kl. 22:00.

På denne uge skal vi tage et nøjere kig på imperativ programmering. Den væsentlige forskel på imperativ og funktionel programmering er, at imperativ programmering arbejder med tilstande, dvs. variable, som kan ændre sig over tid. Som konsekvens heraf kobler man ofte `while` og `for` løkker til imperativ programmering i modsætning til rekurssion. Yderligere bliver håndkøring kompliceret af, at man er nødt til at tilføje en bunke (heap), for at holde øje med tilstandene af variablene over tid.

Denne arbejdsseddels læringsmål er:

- Forklare forskellen på og løse problemer med rekursion, `while` og `for` løkker,
- Skrive programmer med variable, arrays i 1 og 2 dimensioner,
- Håndkøre funktioner med tilstande og `while` og `for` løkker.

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)

8ø0 Consider the factorial-function,

$$n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdot \ldots \cdot n \tag{1}$$

(a) Write a function

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

which uses a `while`-loop, a counter variable, and a local variable to calculate the factorial-function as (**??**).

(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?

8ø1 Perform a trace-by-hand of the following expression `fac 4` using the solution from Exercise **??**,.

8ø2 Consider multiplication tables of the form,

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	19	20
3	3	6	9	12	15	18	21	24	27	30
...										

where the elements of the top row and left column are multiplied and the result is written at their intersection.

In this assignment, you are to work with a function

```
mulTable : n:int -> string
```

which takes 1 argument and returns a string containing the first $1 \leq n \leq 10$ lines in the multiplication table including `<newline>` characters. Each field must be 4 characters wide. The resulting string must be printable with a single`printf "%s"`statement. For example, the call `mulTable 3` must return.

Listing 1: An example of the output from `mulTable`.

```
printf "%s" (mulTable 3);;
        1    2    3    4    5    6    7    8    9   10
    1   1    2    3    4    5    6    7    8    9   10
    2   2    4    6    8   10   12   14   16   18   20
    3   3    6    9   12   15   18   21   24   27   30
```

All entries must be padded with spaces such that the rows and columns are right-aligned. Consider the following sub-assignments:

(a) Create a function with type

```
mulTable : n:int -> string
```

such that it has one and only one value binding to a string, which is the resulting string for $n = 10$, and use indexing to return the relevant tabel for $n \leq 10$. Test `mulTable n` for $n = 1, 2, 3, 10$. The function should return the empty string for values $n < 1$ and $n > 10$.

(b) Create a function with type

```
loopMulTable : n:int -> string
```

such that it uses a local string variable, which is built dynamically using 2 nested `for`-loops and the `sprintf`-function. Test `loopMulTable n` for $n = 1, 2, 3, 10$.

(c) Make a program, which uses the comparison operator for strings, "=", and write a table to the screen with 2 columns: `n`, and the result of comparing the output of `mulTable n` with `loopMulTable n` as `true` or `false`, depending on whether the output is identical or not.

(d) Use `printf "%s"`and `printf "%A"`to print the result of `mulTable`, and explain the difference.

8ø3 Use `Array.init` to make a function `squares: n:int -> int []`, such that the call `squares n` returns the array of the first n square numbers. For example, `squares 5` should return the array `[|1; 4; 9; 16; 25|]`.

8ø4 Write a function `reverseArray : arr:'a [] -> 'a []` using `Array.init` and `Array.length` which returns an array with the elements in the opposite order of `arr`. For eksample, `printfn "%A" (reverseArray [|1..5|])` should write `[|5; 4; 3; 2; 1|]` to the screen.

8ø5 Use `Array2D.init`, `Array2D.length1` and `Array2D.length2` to make the function `transposeArr : 'a [,] -> 'a [,]` which transposes the elements in input.

Afleveringsopgaver (in English)

In this assignment, you are to work with cyclic queues. A cyclic queue is a queue with fixed storage space allocated for it, such as an array of constant length, and two points pointing to the first and last element in the queue. As an example, consider a cyclic queue of length 8 as illustrated in **??**. When the queue maintains the variables `first` and `last`, where `first` points to the position of the element in the front of the queue, and `last` points to the position of the element in the back of the queue, which also is the last element added to the queue. Initially, point pointers are `None`. If the queue is not full, then when enqueueing values to the queue, `last` is cyclicly incremented by 1, and the value is stored in that position. If the queue is not empty, then dequeuing values from the queue, the value at position `first` is returned and lstinlinefirst is cyclicly incremented by 1.

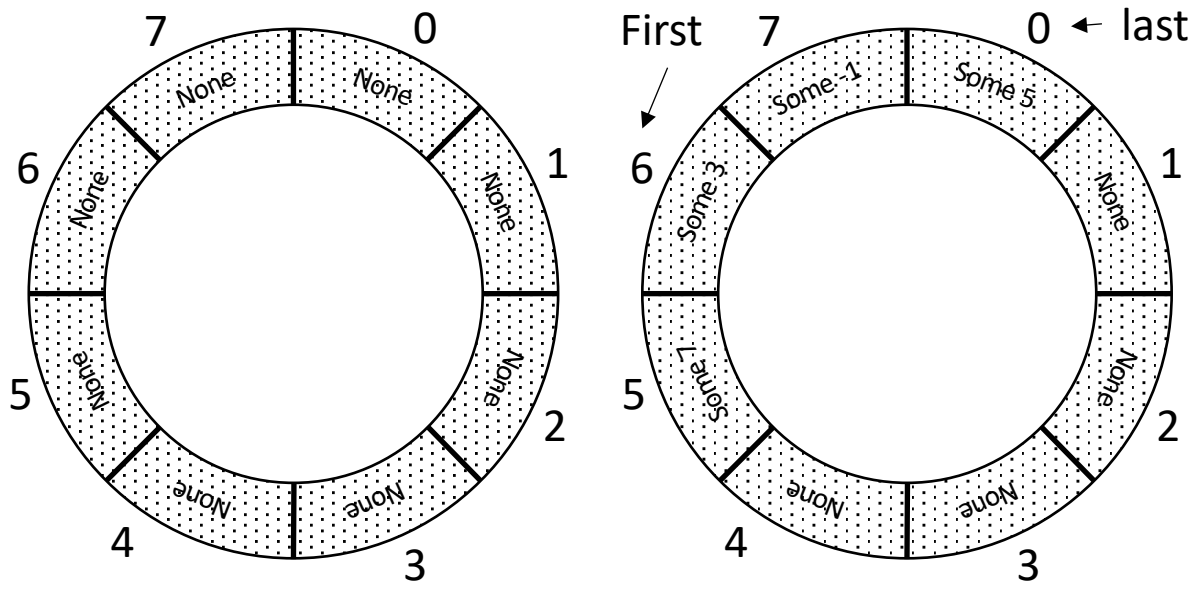
In this assignment, you are to make a module that implements the abstract datatype known as a *cyclic-Queue* for integers using imperative programming and which has the following interface `cyclicQueue.fsi`:

```
module cyclicQueue

type Value = int

/// <summary>Create or clear the cyclic queue</summary>
/// <param name="n">The maximum number of elements</param>
val create: n: int -> unit

/// <summary>Add an element to the end of a queue</summary>
/// <param name="e">an element</param>
/// <returns>True if the queue had space for the element</returns>
val enqueue: e: Value -> bool
```



(a) (b)

Figure 1: A cyclic queue of length 8. Integers on the outside of the figure are position indices. (a) The initial state with no data and where the first and last are also set to `None`. (b) A queue after enqueueing the sequence $7, 3, -1, 5$ and then dequeuing once such that the first element is 3 and the last is 5. The next element to be added will be in position 1.

```
/// <summary>Remove the element in the front position of the
   queue</summary>
/// <returns>The first element in q or None if the queue is
   empty</returns>
val dequeue: unit -> Value option

/// <summary>Check if the queue is empty</summary>
/// <returns>True if the queue is empty</returns>
val isEmpty: unit -> bool

/// <summary>Get the length of the queue</summary>
/// <returns>The number of elements in the queue</returns>
val length: unit -> int

/// <summary>The queue on string form</summary>
/// <returns>A string representing the queue's elements</returns>
val toString: unit -> string
```

8ø6 (a) make an implementation of `cyclicQueue.fsi` called `cyclicQueue.fs`. The implementation must use mutable option integers for `first` and `last`, and a mutable array `q: Value option[]`

(b) Write an application that tests each function. Consider whether you can make your functions cast exceptions.

(c) In comparison with a purely functional implementation of a general queue, what are the advantages and disadvantages of this implementation?

Krav til afleveringen

Afleveringen skal bestå af

- en zip-fil, der hedder `8g.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:

  cyclicQueue.fsi, cyclicQueue.fs, og cyclicQueueApp.fsx

- pdf-dokumentet skal være lavet med LaTeX, benytte `opgave.tex` skabelonen, ganske kort dokumentere din løsning og besvare evt. ikke-programmeringsopgaver.

God fornøjelse.