Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 8 - individuel opgave

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 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)

8ø0 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.

8ø1 Consider the following sum of integers,

$$\sum_{i=1}^{n} i. \tag{1}$$

This assignment has the following sub-assignments:

(a) Write a function

```
sum : n:int -> int
```

which uses the counter value, a local variable (mutable value) `s`, and a `while`-loop to compute the sum $1+2+\cdots+n$ also written in (1). If the function is called with any value smaller than 1, then it is to return the value 0.

(b) By induction one can show that

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, n \geq 0 \tag{2}$$

Make a function

`simpleSum : n:int -> int`

which uses (2) to calculate $1+2+\cdots+n$ and which includes a comment explaining how the expression implemented is related to the mentioned sum.

(c) Write a program, which asks the user for the number n, reads the number from the keyboard, and write the result of `sum n` and `simpleSum n` to the screen.

(d) Make a program, which writes a table to the screen with 3 columns: `n`, `sum n` and `simpleSum n`. The table should have a row for each of $n = 1,2,3,..,10$, and each field must be 4 characters wide. Verify that the two functions calculate identical results.

(e) What is the largest value n that the two sum-functions can correctly calculate the value of? Can the functions be modified, such that they can correctly calculate the sum for larger values of n?

8ø2 Perform a trace-by-hand of the following program

```
let a = 3.0
let b = 4.0
let f x = a * x + b

let x = 2.0
let y = f x
printfn "%A * %A + %A = %A" a 2.0 b y
```

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)

8i0 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 2: 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.

In you are to make a module which implements the abstract datatype known as a *queue* using imperative programming. The queue should implemented as a module with a *single* state `q` containing a *single* queue and the following interface, `queue.fsi`:

```
module queue

/// <summary>Add an element to the end of a queue</summary>
/// <param name="e">an element</param>
val enqueue: e: int -> unit

/// <summary>Remove the element in the front position of the
   queue</summary>
/// <returns>The first element in q</returns>
val dequeue: unit -> int option

/// <summary>Check if the queue is empty</summary>
```

```
/// <returns>True if the que is empty</returns>
val isEmpty: unit -> bool

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

8ø1 (a) make an implementaton of `queue.fsi` called `queue.fs`.

(b) Write an application which tests each function. Consider whether you are able to make your functions cast exceptions.

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

Krav til afleveringen

Afleveringen skal bestå af

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

  queue.fsi, queue.fs, og app.fsx

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

God fornøjelse.