Aflevering af 5i

Peter Asp Hansen (glt832)

23. februar 2023

5i0

Vi indleder med at danne et nyt dotnet projekt kaldet 5i med kommandoen `dotnet new console -lang "F#-o 5i`. Vi har således lavet en 5i.fsproj fil hvori vi kan compile vores signature og implementation filer. Denne fil skal se ud på følgende måde når hele afleveringen er færdig:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net6.0</TargetFramework>
    <RootNamespace>_5i</RootNamespace>
  </PropertyGroup>
  <ItemGroup>fs
    <Compile Include="intQueue.fsi" />
    <Compile Include="intQueue.fs" />
    <Compile Include="safeIntQueue.fsi" />
    <Compile Include="safeIntQueue.fs" />
    <Compile Include="queue.fsi"/>
    <Compile Include="queue.fs"/>
    <Compile Include="testQueues.fs" />
  </ItemGroup>
</Project>
```

Figur 1: `5i.fsproj` filen

Hvis vi placerer os i det rigtige directory og skriver kommandoen `dotnet build` forventer vi det følgende output.

```
Build succeeded.
    0 Warning(s)
    0 Error(s)
```

Figur 2: dotnet build output

Derudover vil vi senere se at kaldes kommandoen `dotnet run`, køres testQueues.fs med moduler defineret i de resterende filer. En vigtig overvejelse er placeringen af filerne. Vi placerer signature filen før den tilsvarende implementation fil. `testQueues.fs` placeres til sidste fordi det er det sidste der køres.

5i1

Vi implementerer nu modulet kaldet IntQueue som skal repræsentere elementerne i en kø. Vi laver en signature fil hvor koden er som følgende:

```
module IntQueue
//definer køen som en heltals liste
type queue = int list
//definer elementerne i køen som heltal
type element = int
//Den tomme kø
val emptyQueue: queue
//Tilføj et element i slutningen af køen
val enqueue: element -> queue -> queue
//Fjern og returner elementet foran køen
//precondition: input i køen må ikke være tom
val dequeue: queue -> element * queue
//Tjek om køen er tom
val isEmpty: queue -> bool
```

Figur 3: IntQueue.fsi fil

I signature filen defineres funktionerne `emptyQueue`, `enqueue`, `dequeue` og `isEmpty`.
Deres funktion er beskrevet med kommentarer i koden. For detaljer om funktionernes opsætning henvises der til implementation filen `intQueue.fs`:

```
module IntQueue
type queue = int list
type element = int

///<summary> emptyQueue definerer en tom kø </summary>
///<returns> en tom liste </returns>
let emptyQueue: queue = []

///<summary> enqueue tager et element og en liste og concatenator dem sammen</summary>
///<param name="a"> et element med som har en heltalsværdi </param>
///<param name="lst"> en liste bestående af heltal </param>
///<returns> en liste hvor a er tilføjet til sidst </returns>
let enqueue (a:element) (lst: queue): queue = lst@[a]

///<summary> dequeue tager en liste og skiller det første element og de resterende
    elementer fra hinanden </summary>
///<param name="lst"> en liste bestående af heltal </param>
///<returns> det først element som et heltal og en heltalsliste </returns>
let dequeue (lst:queue) = lst.Head,lst.Tail

///<summary> isEmpty tager lst og vurderer om den er tom</summary>
///<param name="lst"> en liste bestående af heltal </param>
///<returns> True eller False </returns>
let isEmpty (lst:queue) = lst.IsEmpty
```

Figur 4: IntQueue.fs fil

Funktionerne er defineret vha. list moduler. I dequeue benyttes `lst.Head` der returnerer det første element i listen `lst` og `lst.Tail` som returnerer en liste med de resterende elementer i `lst`. Yderligere bruges også list modulet IsEmpty der returnere en boolean værdi der fortæller om `lst` er tom (`True`) eller ikke tom (`False`).
Som det kan ses i figur 1 bliver `intQueue.fs` og `intQueue.fsi` tilføjet til `5i.fsproj` filen.
Til sidst kopieres følgende kode fra opgavebeskrivelsen og tilføjes til filen `testQueues.fs`

```
module testQueues

//test for intQueue
let intQueueTests () =
    let q0 = IntQueue.emptyQueue
    let emptyTestResult = IntQueue.isEmpty q0
    emptyTestResult
    |> printfn "An empty queue is empty: %A"

    let e1 ,e2 ,e3 = 1,2,3
    let q1 = q0 |> IntQueue.enqueue e1
                |> IntQueue.enqueue e2
                |> IntQueue.enqueue e3
    let nonEmptyTestResult = not (IntQueue.isEmpty q1)

    nonEmptyTestResult
    |> printfn "A queue with elements is not empty: %A"

    let (e,q2) = IntQueue.dequeue q1
    let dequeueTestResult = e = e1
    dequeueTestResult
    |> printfn "First in is first out: %A"

    let allTestResults =
        emptyTestResult &&
        nonEmptyTestResult &&
        dequeueTestResult

    allTestResults
    |> printfn "All IntQueue tests passed: %A"
    // Return the test results as a boolean
    allTestResults
// Run the IntQueue tests
let intQueueTestResults = intQueueTests()
```

Figur 5: testQueues.fs

Denne kode kræver ingen modifikation. Indtastes `dotnet run` i det rigtige directory får vi det følgende output

```
An empty queue is empty: true
A queue with elements is not empty: true
First in is first out: true
All IntQueue tests passed: true
```

Figur 6: dotnet run

Dette er det ønskede resultat og vi kan gå videre til næste opgave.

5i2

I denne opgave fjerner vi vores precondition fra `IntQueue.fsi` og modificerer `IntQueue.fs` således at den returnerer `None` hvis inputtet i køen er tom. Vi opretter en ny fil `safeIntQueue.fs` som indeholder den modificerede kode. Heri modificerer vi funktionen `dequeue`.

```
///<summary> dequeue tager en liste og skiller det første element og de resterende
    elementer fra hinanden </summary>
///<param name="lst"> en liste bestående af heltal </param>
///<returns> det først element som et heltal og en heltalsliste</returns>
let dequeue (lst:queue): (element option)*queue =
    match lst with
        [] -> None, lst
        |_-> Some lst.Head,lst.Tail
```

Figur 7: safeIntQueue.fs

Vi laver en tilsvarende implementation fil kaldet `safeIntQueue.fs` og placerer dem i `5i.fsproj`. Der henvises til figur 1 hvor dette kan ses. Denne kode testes med `testQueues`, hvor der også testes for om `dequeue` returnerer `None`.

```
// Run the SafeIntQueue tests
let Testdequeue = SafeIntQueue.dequeue SafeIntQueue.emptyQueue = (None,[])
printfn "dequeue returns None with input []: %b" Testdequeue
let SafeIntQueueTestResults = safeIntQueueTests ()
```

Figur 8: SafeIntQueue test

Vi får et tilsvarende resultat som i figur 6.

```
Test for safeIntQueue
dequeue returns None with input []: true
An empty queue is empty: true
A queue with elements is not empty: true
First in is first out: true
All safeIntQueue tests passed: true
```

Figur 9: dotnet run 2

Vi har igen opnået det ønskede resultat og begiver os videre.

5i3

Indtil nu har vi været begrænset da alle elementerne i vores kø kun kan være heltal. Vi modificerer derfor koden igen så vores kø kan indeholde flere typer. Her benytter vi syntaxen `<'a>` fra Listing 9.18 eksemplet i FsharpNotes.pdf. Vi opretter og indsætter en modificering af koden fra de forrige opgaver i de to filer `Queue.fs` og `Queue.fsi`. Signature filen kommer til at se ud på følgende måde:

```
module Queue
type queue<'a> = 'a list
type element<'a> = 'a
// check if a queue is empty
val isEmpty: queue<'a> -> bool
// the empty queue
val emptyQueue: queue<'a>
// add an element at the end of a queue
val enqueue: element<'a> -> queue<'a> -> queue<'a>
// remove and return the element at the front of a queue
val dequeue: queue<'a> -> (element<'a> option)*queue<'a>
```

Figur 10: Queue.fsi

Et eksempel på den modificering vi har lavet, er hvor vi definerer typen queue, hvor vi i stedet for at skrive `type queue = int list` nu skriver `type queue<'a> ='a list` og dermed gør typen generisk.

Implementation filen modificeres på en lignende måde

```
1     module Queue
2
3 type queue<'a> = 'a list
4 type element<'a> = 'a
5
6 ///<summary> emptyQueue definerer en tom kø </summary>
7 ///<returns> en tom liste </returns>
8 let emptyQueue: queue<'a> = []
9
10 ///<summary> enqueue tager et element og en liste og concatenator dem sammen</summary>
11 ///<param name="a"> et element med som har er typen <'a> </param>
12 ///<param name="lst"> en liste bestående af <'a> </param>
13 ///<returns> en liste hvor a er tilføjet til sidst </returns>
14 let enqueue (a:element<'a>) (lst: queue<'a>): queue<'a> = lst@[a]
15
16 ///<summary> isEmpty tager lst og vurderer om den er tom</summary>
17 ///<param name="lst"> en liste bestående af elementer af typen <'a></param>
18 ///<returns> True eller False </returns>
19 let isEmpty (lst:queue<'a>) = lst.IsEmpty
20
21 ///<summary> dequeue tager en liste og skiller det første element og de resterende
      elementer fra hinanden </summary>
22 ///<param name="lst"> en liste bestående af elementer af en given type <'a> </param>
23 ///<returns> det først element og en liste</returns>
24 let dequeue (lst:queue<'a>): (element<'a> option)*queue<'a> =
25     match lst with
26         [] -> None, lst
27         |_-> Some lst.Head,lst.Tail
28
```

Figur 11: Queue.fs

Bemærk at `queue` og `element` blot er blevet udskiftet med `queue<'a>` og `element<'a>`. Hertil følger at vi har modificeret koden i testQueues.fs således, at den også tager en generisk type. Denne ændring sker i linje 10 i figur 5 hvor `let e1,e2,e3 = 1,2,3` ændres til `let e1,e2,e3=a,b,c`. Yderligere køres der tests for typerne int,float og string.

```
1 // Run the Queue tests
2 printfn "Test for Queue with int:"
3 let QueueTestResultsInt = (QueueTests 1 2 3)
4 printfn "Test for Queue with float:"
5 let QueueTestResultsFloat = (QueueTests 1.0 2.0 3.0)
6 printfn "Test for Queue with string:"
7 let QueueTestResultsString = (QueueTests "a" "b" "c")
```

Figur 12: Tests for int, float og string

Denne kode giver os det følgende output:

```
Test for Queue with int
An empty queue is empty: true
A queue with elements is not empty: true
First in is first out: true
All Queue tests passed: true

Test for Queue with float
An empty queue is empty: true
A queue with elements is not empty: true
First in is first out: true
All Queue tests passed: true

Test for Queue with string
An empty queue is empty: true
A queue with elements is not empty: true
First in is first out: true
All Queue tests passed: true
```

Figur 13: Queue test output

Således er vi nu i mål med afleveringen.