Learning to Program with F#
Exercises
Department of Computer Science
University of Copenhagen

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

November 21, 2022

0.1 Queue

0.1.1 Teacher's guide

Emne rekursion, grafik og winforms

Sværhedsgrad Middel

0.1.2 Introduction

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
```

0.1.3 Exercise(s)

0.1.3.1: (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?