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

Jon Sporring, Martin Elsman, Torben Mogensen, Christina Lioma

October 21, 2022

0.1 Abstract list

0.1.1 Teacher's guide

0.1.2 Introduction

0.1.3 Exercise(s)

0.1.3.1: A linked list is an abstract datatype that is built into `F#`.

In this exercise you will create your own implementations of a linked list, one only supporting `int` and a generic linked list that can be used with any datatype, just like the built-in `F#` lists.

(a) Copy the following signature into `intLinkedList.fsi` and write a corresponding implementation in `intLinkedList.fs`

```
module IntLinkedList
type intLinkedList = Nil | Cons of int * intLinkedList
val head : intLinkedList -> int
val tail : intLinkedList -> intLinkedList
val isEmpty : intLinkedList -> bool
val length : intLinkedList -> int
val add : int -> intLinkedList -> intLinkedList
```

Write a small test program to ensure your implementation works as expected. You can take inspiration from the following listing:

```
open IntLinkedList
let emptyList = Nil
let l1 = Cons (1, Nil)
let l2 = Cons (1, Cons (2, Nil))
let l3 = add 3 l2
isEmpty emptyList |> printfn "Empty list is empty: %A"
isEmpty l1 |> not |> printfn "Non-empty list is not empty: %A"
head l1 = 1 |> printfn "head gives the first element: %A"
tail l1 = Nil |> printfn "tail gives the rest of the list: %A"
length l3 = 3 |> printfn "l3 has length 3: %A"
```

(b) In the previous sub-exercise the linked list is restricted to the type `int`. In this sub-exercise you should construct a generic linked list module.

Copy and finish the following signature in `linkedList.fsi` and write a corresponding implementation in `linkedList.fs`.

```
module LinkedList
type LinkedList<'a> = Nil | Cons of 'a * intLinkedList<'a>
val head : LinkedList<'a> -> 'a
val tail : ?? // Fill in yourself
val isEmpty : ?? // Fill in yourself
val length :  ?? // Fill in yourself
val add :  ?? // Fill in yourself
```

Write a small test-program showing you can construct linked lists of all types. You should be able to reuse `all` of your test from the previous sub-exercise and thus create linked lists of `LinkedList<int>`, as well as the following:

```
open LinkedList
let emptyList = Nil
let l1Float = add 2.0 emptyList |> add 3.14 // A float list
let l1String = add "Linked lists are cool!" emptyList
let l2String = add "What is cool?" l1String
// A list of int lists
let intLstLst = add l1 emptyList |> add l2 |> add emptyList
// a list of string lists
let strLstLst = add l1String emptyList |> add l2String
```

(c) Implement `fold` for your linked list module, similar to `List.fold`.

(d) Implement `foldBack` for your linked list module, similar to `List.foldBack`.

(e) Implement `map` for your linked list module, similar to `List.map`.