- TD -
Higher-order Functions

1. Binary string transmitter

We want to define functions allowing the transmission of strings over a channel accepting only the transmission of binary digits. For this, we should define a function which encodes lists of strings into lists of bits and another one decoding the other way out.

This exercise is strongly inspired from the one proposed in Chapter 7 of the book Programming in Haskell, Graham Hutton, Cambridge University Press, 2016.

1.1. Auxiliary functions

Define a function `iterate` which produces an infinite list by applying a function an increasing number of times to a value

```
iterate :: (a -> a) -> a -> [a]
```

For example, `iterate (++"a") "a"` produces `["a", "aa", "aaa", "aaaa", "aaaaa", …]`.

Define a function `fill xs v n` which adds as many v at the end of the list xs (supposed to have a length smaller than n) to get a list of length n:

```
fill :: [a] -> a -> Int -> [a]
```

For example, `fill [1..2] 0 4` produces `[1,2,0,0]`.

Define a function `chop n xs` which chops a list `xs` into lists of length `n` (the last one could have a length smaller than `n`) :

```
chop :: Int -> [a] -> [[a]]
```

For example, `chop 2 [1,2,0,0]` produces `[[1,2],[0,0]]`.

1.2. Binary to Integer

First, we consider a type for bits:

```
type Bit = Int
```

Define the function `bin2int` which returns the integer encoded by a list of bits:

```
bin2int :: [Bit] -> Int
```

Thus, `bin2int [1,1,0,1]` produces `13` = 2^3 x `1` + 2^2 x `1` + 2^1 x `0` + 2^0 x `1`.

Give two versions for this function:
- One based on list comprehensions.
  Hint: you can use iterate.
- One using fold.
  Hint: 2^0 x 1 + 2^1 x 0 + 2^2 x 1 + 2^3 x 1
  = 1 + 2 x (0 + 2 x (1 + 2 x 1))

1.3. Integer to Binary

Define the function `int2bin` which returns the list of bits representing an integer:

```
int2bin :: Int -> [Bit]
```

For example, `int2bin 13` produces `[1,1,0,1]`. For this, we note that

13 div 2 = 6 remainder 1
6 div 2 = 3 remainder 0
3 div 2 = 1 remainder 1
1 div 2 = 0 remainder 1

and the sequence of remainders, from the last to the first, 1101, provides the binary representation of the integer 13.

Modify the function to produce bytes, i.e. 8 bit lists. For example, `int2bin' 13` produces `[0,0,0,0,1,1,0,1]`.

1.4. Transmission

Define the function `encode` which encodes a string, i.e. a list of characters, into a list of bits by converting each character in the list into its ASCII code, transforming it into a byte, and concatenating all bytes. For example, `encode "JB"` produces `[0,1,0,0,1,0,1,0,0,1,0,0,0,0,1,0]`.
Hint: you can use the function `ord` to convert `Char` into `Int` (`import Data.Char`).

Define the function `decode` which decodes a a list of bits by chopping it into bytes, i.e. 8 bits lists, transforming them into integers, and converting into the corresponding ASCII code.
Hint: you can use the function `chr` to convert `Int` into `Char`.

We can now use the function transmit that simulates the transmission of a string of characters over a communication channel:

```
transmit :: String -> String
transmit = decode . channel . encode
```

The perfect communication channel can be modelled using the identity function

```
channel :: [Bit]->[Bit]
channel = id
```

and in this case `transmit "good work"` produces `"good work"`.

1.5. Error detection

Transmission errors, like missing or changed bits during communication, can arise during the communication. Modify the encoder/decoder programs to allow the detection of such communication errors using parity bits. For this, each byte produced as result of the encoding of a character is extended with a parity bit, set to one if the number contains an odd number of ones, and to zero otherwise. When decoding the incoming list of bits, each nine-bit binary number is checked to ensure that its parity bit is correct and the parity bit is discarded when this is the case; a parity error reported otherwise.

Hint: you can use the function `error :: String -> a` to terminate the evaluation and display an error message.

1.6. Faulty transmissions

Modify the function encoding the channel to simulate faulty transmissions and check that the encoder/decoder programs modified to detect such problems work correctly.

1.7. Alternative definitions

A higher-order function `unfold` that encapsulates a simple pattern of recursion for producing a list can be defined as follows:

```
unfold :: (t -> Bool) -> (t -> a) -> (t -> t) -> t -> [a]
unfold p h t x
  | p x = []
  | otherwise = h x : unfold p h t (t x)
```

The function `unfold p h t` produces the empty list if the predicate p is true on the argument, and otherwise produces a non-empty list by applying the function h to give the head, and the function t to generate another argument that is recursively processed in the same way to produce the tail of the list.

Use the function unfold to give alternative, more compact, definition for the functions:

- `int2bin`
- `chop`
- `map`
- `iterate`

Hints

For `bin2int`, you could define a function `bin2intR` which considers that the bit list is given in reversed order, i.e. given `[1,0,1,1]` produces 2^0 x `1` + 2^1 x `0` + 2^2 x `1` + 2^3 x `1` = `13`.