CO2008@UoL - Worksheet 4: *Datatypes*

Template file:	Worksheet4.hs
Labs:	Friday 8 and 15 March 2019
Hand-in:	18.00 hr on 24 Sunday March 2019
Topics:	Algebraic and recursive types. Trees, paths and errors.

1. A pack of playing cards contains 52 cards. Each card has a 'value' which is taken to be an element of the type

   ```
   data Value = Two|Three|Four|Five|Six|Seven|Eight|Nine|Ten|J|Q|K|A
   ```

   and it has a 'suite' which is taken to be an element of the type

   ```
       data Suite  = Hearts | Spades  | Diamonds| Clubs
   ```

   A card is thus an element of

   ```
       type Card = (Value, Suite)
   ```

   (a) Define a show function for Value that transfers the values into the following strings: "A", "2", ..., "10", "J", "Q", "K",

   (b) Define a show function for Suite that transfers the elements of Suite into the strings: "H", "S", "C", "D"

   (c) (unassessed) Is it possible to write a show function for Card that would transform (A,Heart) into the string AH.

   (d) Give a concise definition of a value `pack::[Card]` containing all of the possible playing cards in some order. Use list comprehension.

   (e) There are two colours of playing card

   ```
       data Colour = Red | Black
   ```

   A card is Red if its suite is either Diamonds or Hearts and is Black otherwise. Write a function to determine the colour of a card.

   (f) A common way to shuffle a pack of cards is to repeatedly split the pack roughly in the middle and then to interleave the two portions. Write a function `split::Int->[a]-> Error ([a],[a])` so that `split n` divides a list in two at the point just after the `n`-th element (we start counting from `n=1`). For instance,

   ```
    split 0 [1,2, 3,4,5] = Ok ([], [1,2,3,4,5])
    split 2 [1,2, 3,4,5] = Ok ([1,2], [3,4,5])
   ```

   Give an error in case the number is negative or larger than the length of the list. For instance,

```
split 8 [1,2, 3,4,5] = Fail
split (-5) [1,2, 3,4,5] = Fail
```

Write a second function to interleave two lists of type [a], possibly of different lengths. For instance, interleaving two list of integers looks like this:

```
interleave [1,2,3] [4,5,6,7,8,9] = [1,4,2,5,3,6,7,8,9].
```

(g) A shuffle of the pack is specified by giving a list of integers. For example, the list `standard` below corresponds to the shuffle in which the pack is split after the 23rd card, interleaved, split again after the 26th card, interleaved, and so on.

```
standard :: [Int]
standard = [23,26,25,31,19,27]
```

Write a function `shuffle` which can be used on any list `xs` to calculate the effect of shuffling the list according to a list of integers.

Use the functions `split` and `interleave` defined in the previous part. Give an error in case `split` gives an error, i.e. give an error in case there is a number which is negative or larger than the length of the list `xs`.

2. A binary tree can be used as a database. Here, the leaves of a tree are either `ND` indicating *no data*, or `Data d` where `d` is a data item.

```
data Btree a = ND | Data a |  Branch (Btree a) (Btree a)
```

One can give a *path* to a leaf by giving a list such as `[L,R,L]` which indicates the leaf one arrives at by moving left, right, left from the root of the tree (of course, there may be no such leaf).

```
data Dir = L | R

type Path =  [Dir]
```

(a) Define `extract` which given a path and a binary tree, outputs the data at the end of the path, and gives an error value when the path does not match any data.

(b) Define `add`, whose three inputs are some data, a path, and a binary tree. The output consists of the binary tree, modified to include the data item at the end of the path. In more detail, if the input path ends in a leaf node of the form `ND` the tree is extended to contain the new data. If the path leads to a branching node or a leaf node of the form `Data d` an error value is given.

(c) Suppose the tree holds data of type `a`. Define the function `findpath`, which given a function `f`, some data `x` and a tree `t`, returns the lists of paths (possible empty) in `t` to the nodes of the form `Node d` where `f d` is equal to `x`.