Exercise 1. Lists

Multisets, or bags, can be represented as list of pairs \((x, n)\) where \(n\) indicates the number of occurrences of \(x\) in the multiset.

\[
\text{type } \text{Bag } a = [(a, \text{Int})]
\]

For the following exercises you can assume the following properties of the bag representation. \textbf{But note}: Your function definitions have to maintain these properties for any multiset they produce!

1. Each element \(x\) occurs in at most one pair in the list.
2. Each element that occurs in a pair has a positive counter.

As an example consider the multiset \(\{2, 3, 3, 5, 7, 7, 7, 8\}\), which has the following representation (among others).

\[
[(5,1), (7,3), (2,1), (3,2), (8,1)]
\]

Note that the order of elements is not fixed. In particular, we cannot assume that the elements are sorted. Thus, the above list representation is just one example of several possible.

(a) Define two functions \(\text{ins}\) and \(\text{del}\) that insert and remove, respectively, an element into/from from a multiset.

\[
\text{ins} :: \text{Eq } a => a \rightarrow \text{ Bag } a \rightarrow \text{ Bag } a
\]
\[
\text{del} :: \text{Eq } a => a \rightarrow \text{ Bag } a \rightarrow \text{ Bag } a
\]

(Note: The class constraint "\text{Eq } a \Rightarrow" restricts the element type \(a\) to those types that allow the comparison of elements for equality with \(==\).)

(b) Define a function \(\text{bag}\) that takes a list of values and produces a multiset representation.

\[
\text{bag} :: \text{Eq } a => [a] \rightarrow \text{ Bag } a
\]

For example, with \(xs = [7, 3, 8, 7, 3, 2, 7, 5]\) we get the following result.

\[
> \text{bag } xs
\]
\[
[(5,1), (7,3), (2,1), (3,2), (8,1)]
\]

(Note: You probably should make use of the function \(\text{ins}\) defined in the previous part.)
(c) Define a function `subbag` that determines whether or not its first argument bag is contained in the second.

```haskell
subbag :: Eq a => Bag a -> Bag a -> Bool
```

Note that a bag `b` is contained in a bag `b'` if every element that occurs `n` times in `b` occurs also at least `n` times in `b'`.

(d) Define a function `isbag` that computes the intersection of two multisets.

```haskell
isbag :: Eq a => Bag a -> Bag a -> Bag a
```

(e) Define a function `size` that computes the number of elements contained in a bag.

```haskell
size :: Bag a -> Int
```

**Exercise 2. Histogram**

Continuing with the bag data structure from the previous exercise, define a function `hist` that produces a histogram for an integer bag as a `String` value. The function `ph`, which is defined in the template file and uses the definition of `hist`, can be used to print the histogram.

```haskell
hist :: Bag a -> String
```

A histogram is a two-dimensional diagram that plots elements along the x axis. It prints for each element that occurs `n` times in the bag a column of `n` Xs. For example, for `b = [(5,1),(7,3),(2,1),(3,2),(8,1)]` we obtain the following histogram.

```haskell
> ph b
3 X
2 X X
1 XX X XX
12345678
```

Remember that a string is a list of characters, which means that you can employ all list operations (such as, `++`, `map`, etc.) to manipulate strings. The newline character `\n` can be used within strings. For example, we have.

```haskell
> putStrLn ("line 1\nline 2" ++ "\n" ++ "line 3")
line 1
line 2
line 3
```

For simplicity, the borders of the histogram have to work only for bags with the following properties. But the core of your implementation should work for any bag.

- The bag contains only elements in the range 1 to 9.
- Each element occurs at most nine times

**Hint:** It is best to decompose this problem into a series of subtasks. For example, you can decompose the bag into different parts based on the occurrence of elements and then create the histogram line by line.
Exercise 3. Data Types

Here is the definition of a data type for representing a few basic shapes. A figure is a collection of shapes.

```haskell
type Number = Int

type Point = (Number,Number)
type Length = Number

data Shape = Pt Point
  | Circle Point Length
  | Rect Point Length Length
  deriving Show

type Figure = [Shape]

type BBox = (Point,Point)
```

The type `BBox` represents bounding boxes of objects by the points of the lower-left and upper-right hand corners of the smallest enclosing rectangle.

(a) Define the functions `width`, `bbox`, and `minX` that compute the width, bounding box, and minimum x coordinate of a shape, respectively.

```haskell
width :: Shape -> Length
bbox :: Shape -> BBox
minX :: Shape -> Number
```

For example, the widths, bounding boxes, and minimum x coordinates of the shapes in the following figure f are as shown.

```haskell
f = [Pt (4,4), Circle (5,5) 3, Rect (3,3) 7 2]

> map width f
[0,6,7]

> map bbox f
[((4,4),(4,4)),((2,2),(8,8)),((3,3),(10,5))]

> map minX f
[4,2,3]
```

(b) Define a function `move` that moves the position of a shape by a vector given by a point as its second argument.

```haskell
move :: Shape -> Point -> Shape
```
It is probably a good idea to define and use an auxiliary function \( \text{addPt} :: \text{Point} \rightarrow \text{Point} \rightarrow \text{Point} \), which adds two points component wise.

(c) Define a function \( \text{alignLeft} \) that transforms one figure into another one in which all shapes have the same \( \text{minX} \) coordinate but are otherwise unchanged.

\[
\text{alignLeft} :: \text{Figure} \rightarrow \text{Figure}
\]

*Note:* It might be helpful to define an auxiliary function \( \text{moveToX} :: \text{Number} \rightarrow \text{Shape} \rightarrow \text{Shape} \) that changes a shape's position so that its \( \text{minX} \) coordinate is equal to the number given as first argument.

(d) Define a function \( \text{inside} \) that checks whether one shape is inside of another one, that is, whether the area covered by the first shape is also covered by the second shape.

\[
\text{inside} :: \text{Shape} \rightarrow \text{Shape} \rightarrow \text{Bool}
\]

*Hint:* Think about what one shape being inside another means for the bounding boxes of both shapes. Note that this remark is meant to help with some cases, but it doesn't solve all.