Homework 1

Functional Programming (ITI0212)

due: 2021.02.22

Place your solutions in a module named `Homework1` in a file with path `homework/Homework1.idr` within a repository called `iti0212-2021` on the TalTech GitLab server (`https://gitlab.cs.ttu.ee/`). Your solutions will be pulled automatically for marking. At the start of the file include a comment containing your name and the Idris version you are using. Precede each problem's solution with a comment specifying the problem number.

Problem 1
The *Ackermann function* is a famously fast-growing total computable function with the following type:

```
ack  :  Nat -> Nat -> Nat
```

and recursively defined by:

$$\text{ack } m\ n = \begin{cases} n+1 & \text{if } m = 0 \\ \text{ack } (m-1)\ 1 & \text{if } m \neq 0 \text{ and } n = 0 \\ \text{ack } (m-1)\ (\text{ack } m\ (n-1)) & \text{otherwise} \end{cases}$$

Write the Ackermann function in Idris using pattern matching. Make sure Idris agrees that your function is total and confirm that it returns correct results for some low argument values according to `https://en.wikipedia.org/wiki/Ackermann_function#Table_of_values`.

Problem 2
Write total functions with each of the following parameterized types:

```
diag   :  a -> Pair a a
anyway :  Either a a -> a
assocr :  Pair (Pair a b) c -> Pair a (Pair b c)
distl  :  Pair a (Either b c) -> Either (Pair a b) (Pair a c)
```

Problem 3
Write the polymorphic function that consolidates a `List` of `Maybe` elements into a `Maybe List` of elements, which returns a `Just` list just in case the argument is a list of `Just`s. For example:

```
> consolidate []
Just []
> consolidate [Just 1 , Just 2 , Just 3]
Just [1, 2, 3]
> consolidate [Just "A" , Nothing , Just "C"]
Nothing
```

Problem 4
Write a higher-order function that uses a given function to transform the element at the specified index of a list:

```
transform  :  (f : a -> a) -> (index : Nat) -> List a -> List a
```

If the index is out-of-bounds for the list then your function should behave like the identity function. For example:

```
> transform S 0 [1 , 2 , 3]
[2, 2, 3]
> transform S 1 [1 , 2 , 3]
[1, 3, 3]
> transform S 2 [1 , 2 , 3]
[1, 2, 4]
> transform S 3 [1 , 2 , 3]
[1, 2, 3]
```

Problem 5
Write a function that capitalizes the first character of each word of a string. For example:

```
> titlecase "it was the best of times it was the worst of times"
"It Was The Best Of Times It Was The Worst Of Times"
```

You may assume that the words are separated by whitespace. The following standard library functions may be helpful:

- `words : String -> List String` and `unwords : List String -> String`,
- `unpack : String -> List Char` and `pack : List Char -> String`,
- `toUpper : Char -> Char`.

The functions `toUpper`, `pack` and `unpack` are in `Prelude`, which is imported automatically by default. Depending on the version of Idris you are using, `words` and `unwords` may be located either in the prelude or in the library `Data.String` (misnamed "`Data.Strings`" in some versions), which you will need to import explicitly in order to use.

tip: you can write this as a one-liner using your function from problem 4, function composition, and the prelude function `map : (a -> b) -> List a -> List b`, which applies the given function to each element of the given list.

Problem 6
In Idris we can define a type of m-by-n matrices as follows:

```
Matrix  :  Nat -> Nat -> Type -> Type
Matrix m n t  =  Vect m (Vect n t)
```

By this definition, a matrix is a vector of `m` rows, each of which is a vector of `n` cells, each of type `t`. This problem requires that you import `Data.Vect` and include this definition of the type of m-by-n matrices in your Idris file.

Recall that in order to add two matrices of integers they must have the same dimensions as each other, and moreover the result of the the addition will again have the same dimensions. We can encode this property of matrix addition in the type of the addition function as follows:

```
add  :  Matrix m n Integer -> Matrix m n Integer -> Matrix m n Integer
```

Write this function that adds m-by-n matrices.