Department of Information and Computing Sciences
Utrecht University

Refresh Functional Programming for Languages and Compilers

Practice exercises (Should not be handed in)

1. A text can be stored as a `String`, but also as a list of words, so as a `[String]` type. In the latter case the spaces between words don't need to be stored. It differs from case to case which one is more useful, so therefore it is good to have conversion functions.

In the standard Prelude these are defined:

```
words   :: String  -> [String]
unwords :: [String] ->  String
```

Try to write these functions yourselves, in the following ways:

- `unwords` using direct recursion
- `unwords` using `foldr`
- `words` using recursion and helper functions like `takeWhile`
- `words` using `foldr` (It is suprising that this can be done. It is hard to come up with the correct solution, but once written you will see the beauty of it. Hint: the operator that handles a single letter must handle a space in a different way than another character.)

2. There are two types of fold-functions: `foldr` and `foldl`. They work as follows:

```
foldr (+) e [a,b,c,d] = a + (b + (c + (d + e)))
foldl (+) e [a,b,c,d] = (((e + a) + b) + c) + d
```

Give the type and the recursive definition of `foldl` (and compare it with that of `foldr`).

3. Write a function that converts a list of digits to the corresponding number, for example:

```
f [1,3,6] = 136
```

Use `foldr` or `foldl` for this.

After this, create a function that converts a `String` (assume that it contains only digit-symbols) to an `Int`.

4. You are given the following type for binary trees with information in the leaves:

```
data Tree a = Bin (Tree a) (Tree a)
            | Tip a
```

Write a function `information :: Tree a -> [a]` returning a list of all information from the leaves.

5. Write a function `pack :: Tree Int -> String` that converts the tree to a `String` representation. The `String` representation should not only contain the numbers from the leaves, but also represent the structure of the tree: for every `Bin`-node the children must be separated by a comma, and the full representation of the node must be enclosed in curly brackets.

6 (*Hard, but very relevant for 'parsing'*)**.** Write a function doing the inverse as that in the previous exercise: `unpack :: String -> Tree Int`.

Hint: first write a function that solves a slightly harder problem: next to the resulting `Tree` this function should also return the remaining part of the input `String`. In other words, there can be 'garbage' in the input `String` after the tree. When you have defined this function then the `unpack` function should not be hard to define, and the help-function will be of great use.