Question 1 [13 marks]

When decompressing bit strings, the mapping between bit strings and the output string can be represented as a tree. A bit string is represented as a list of 0's and 1's.

Consider the following code, where `DecTree t` is a tree that is either the output value, or a split where the left branch is selected by 0 and the right branch by 1.

```
data DecTree d  = Val d
           | Spl (DecTree d) (DecTree d)

instance (Show t) => Show (DecTree t) where
   show a = showb a []
showb :: Show a => DecTree a -> [Char] -> [Char]
showb (Val t) p = p++":"++show t
showb (Spl lt rt) p = (showb lt (p++"0"))++", "++(showb rt (p++"1"))

stree = Spl (Val "fun") (Spl (Val "is") (Val "no"))
```

The following shows an interaction with ghci:

```
ghci> stree
0:"fun", 10:"is", 11:"no"
```

(a) [2 marks] What does `d` represent in the `data` definition?

(b) [3 marks] Explain in English (suitable for one of your peers who is just starting to learn Haskell) the point of the line that starts with `instance` and the line that follows. (Do not try to explain what `showb` does.)

(c) [8 marks] Finish the function `decode1` which takes a tree and a list of 0's and 1's and returns a pair of the leaf value reached and the rest of the list:

```
ghci> decode1 stree [1,1,0,1,0,1,1,0]
("no",[0,1,0,1,1,0])
ghci> decode1 stree [0,1,0,1,1,0]
("fun",[1,0,1,1,0])
```

```
decode1 ::                                    -> [Int] -> (u, [Int])

decode1                                     c = (v, c)

decode1 (Spl lt _)                                    = decode1 lt c

decode1 (Spl _ rt) (1:c) =
```

Question 2 [12 marks]

David used `decode1` to implement `decode`, which decodes the whole bit stream (lazily), with definition:

```
decode :: DecTree u -> [Int] ->  [u]
decode _ []  = []
decode ct lst
    = (val : decode ct rc)
    where (val, rc) = decode1 ct lst
```

(a) [3 marks] David was unhappy with the program because it sometimes gave a runtime error (even when provided with a list containing only 0's and 1's). Why it did give a runtime error? (Specify which function gives an error and why.)

(b) [3 marks] David thought of fixing the run time error by making the output of `decode1` to be of type `Maybe (u, [Int])`, and the output of `decode` to be of type `Maybe [u]`. However he rejected that idea as he thought it would be very inefficient for very large files (long binary strings) when we might only want to use the first part of the decoded result. Why might it be more inefficient when the output type includes the `Maybe`?

(c) [3 marks] David decided instead to add

```
decode1 _ [] = ("",[])
```

but got the error

```
mid2test.hs:19:17: error:
  • Couldn't match expected type ‘u’ with actual type ‘String’
   |
19 | decode1 _ [] = ("",[])
   |                 ^^
```

Why did he get the error? How could he fix the error so that this definition would work? What is wrong with that fix?

(d) [3 marks] David instead thought that he would avoid the error by treating the list as though it was padded by 0's at the end, so that in the above case, a [1] would be treated as [1,0], and so give `"is"`. Show what can be added where to implement this. Hint: it can be done by adding one line.

Question 3 [11 marks]

Consider the definition (same as before but using the default version of `show`):

```
data DecTree d  = Val d
           | Spl (DecTree d) (DecTree d)
     deriving Show
stree = Spl (Val "fun") (Spl (Val "is") (Val "no"))
```

Do not use any built-in functions in your answer.

(a) [5 marks] Reminder: `map` for lists has the type:

```
map :: (a -> b) -> [a] -> [b]
```

Define a version of `map`, called `maptree`, for trees defined using the constuctors above, with the following behavior:

```
ghci> maptree length stree
Spl (Val 3) (Spl (Val 2) (Val 2))
ghci> maptree reverse stree
Spl (Val "nuf") (Spl (Val "si") (Val "on"))
```

Give the type declaration and a definition of `maptree`:

(b) [6 marks] Recall that my version of `foldr` for lists has type:

```
myfoldr :: (t1 -> t2 -> t2) -> t2 -> [t1] -> t2
```

Define a version of `foldr`, called `treefold`, that folds the elements at the leaves of such trees, with the following behavior:

```
ghci> treefold (:) [] stree
["fun","is","no"]
ghci> treefold (++) "good" stree
"funisnogood"
ghci> treefold (:) [4,5] (Val 3)
[3,4,5]
```

Give the type of `treefold`:

```
treefold ::
```

Give a definition of `treefold`:

```
treefold f b (Val t)  =

treefold f b (Spl lt rt) =
```

Question 4 [9 marks]

(a) [3 marks] In Haskell the function . is defined by

```
(f . g) x = f (g x)
```

and `length` gives the length of a list. Suppose a new student had just heard about reductions, but was confused by

```
ghci> ((+).length) "abc" 4
7
```

Show the reductions that give this result (assume that `(+)` and `length` compute their answer in a single reduction):

(b) [4 marks] Give two (different) advantages of having no side effects.

Advantage 1:

Advantage 2:

(c) [2 marks] Haskell can infer the type of a functions that can compile. Why should a programmer provide their own type declarations?