# DM580 – EXERCISE SHEET #6

(1) Write a function `safetail :: [a] -> Maybe [a]` that works just like `tail` except it returns `Nothing` instead of producing an error when the input is invalid.

(2) Consider the type of cardinal directions below.

```
data Direction = North
               | South
               | East
               | West
```

  (a) Write type class instances of `Eq` and `Show` for `Direction` (*just this once* without using derived instances).

  (b) Cardinal directions can be combined to form the *intercardinal* directions (e.g., southeast), but only if they are adjacent (if they are not, we instead get nonsense such as *northsouth*). Write a function `adjacent :: Direction -> Direction -> Bool` that checks if two cardinal directions are adjacent.

(3) Consider the type of binary trees below.

```
data Tree a = Leaf a
            | Node (Tree a) (Tree a)
```

  (a) A binary tree is *balanced* if it is either a leaf, or the number of leaves in its left and right subtrees differ by at most 1. Write a function `balanced :: Tree a -> Bool` that checks if a tree is balanced or not.

  (b) Write a function `visit :: Tree a -> [a]` that finds all values stored at the leaves of a given tree.

(4) Without consulting the Haskell Prelude, complete the type class instances below.

```
instance Eq a => Eq (Maybe a) where
  -- Complete me!

instance Eq a => Eq [a] where
  -- Complete me!
```

(5) In propositional logic, a *literal* refers to either a variable or the negation of a variable. A *disjunctive clause* is a formula consisting of any number of literals separated by $\vee$ (or). A formula is said to be in *conjunctive normal form* (CNF) iff it consists of any number of disjunctive clauses separated by $\wedge$ (and). For example, the following formulas are in conjunctive normal form:

$$(a \vee \neg b) \wedge (c \vee \neg c \vee \neg d) \qquad (a \vee b) \wedge \neg c \qquad a \vee \neg a \vee \neg b \qquad a \wedge \neg a$$

  (a) Define a type `CNF` that allows one to express arbitrary formulas in conjunctive normal form. It must be possible to use any `String` as the name of a variable.

  (b) Ensure that `CNF` is an instance of the type classes `Eq` and `Show`.

  (c) Define a type `Env` of *environments* consisting of assignments of names of variables to truth values (i.e., Boolean values).

  (d) Write a function `env :: Env -> String -> Bool` that returns the truth value of a variable in a given environment.

  (e) Write a function `eval :: Env -> CNF -> Bool` that computes the truth value of a CNF-formula (with truth values of variables given by the supplied environment).

  (f) Write a function `vars :: CNF -> [String]` that computes the names of all variables occuring in a given CNF-formula.

  (g) Write a function `envs :: CNF -> [Env]` that computes all possible environments for all variables occuring in the given CNF-formula.

  (h) A propositional logic formula is *satisfiable* iff there exists an assignment of variables to truth values (i.e., an environment) that makes the formula true. Write a function `sat :: CNF -> Bool` that computes if a given CNF-formula is satisfiable or not.
  *Hint:* Use generate-and-test and your helper functions above.

(6) *Challenge:* A propositional logic formula is *valid* if it is always true no matter how you assign truth values to its variables. It can be shown that a CNF-formula is valid iff each of its clauses contains some literal and its negation. Write a function `valid :: CNF -> Bool` that uses this property to check if a given CNF-formula is valid or not.