```
-- Thomas Alessandro Buse 192959, Übung 9, Gruppe 17
-- Paul Rüssmann 196683
{-# LANGUAGE TypeFamilies #-}

--Aufgabe 9.1
import Examples
fib 0 = 1
fib 1 = 1
fib n = fib(n-1) + fib(n-2)
--Aufgabe 9.2 a)

isCyclic :: Eq a => Graph a -> Bool
isCyclic g = loop $ graph2Rel $ closureF g
        where loop [] = False
                loop ((a,b):xs) = elem (b,a) xs || loop xs
--Aufgabe 9.2 b)

depthFirst :: Eq a => a -> Graph a -> [a]
depthFirst start g = dfs [start] []
        where dfs [] visited = visited dfs (a:as) visited| elem a visited = dfs as visited
        | otherwise = dfs ((reachables g a)++as) (visited++[a])

--Aufgabe 9.3 a)

-- class C f where
-- comp :: f b c -> f a b -> f a c
-- f ::(* -> * -> *)

--Aufgabe 9.3 b)

-- data T f g = t (f String Int) (g Bool)
-- T :: (* -> * -> *) -> (* -> *) -> *
```

```
--Aufgabe 9.4 a)

class Listable l where
type Item l :: *
toList :: l -> [Item l]

data Colist a = Colist {split :: Maybe (a,Colist a)}
nil :: Colist a
nil = Colist Nothing

instance Listable (Colist a) where
type Item (Colist a) = a
toList (Colist Nothing) = []
toList (Colist(Just(a,b))) = a : (toList b)

--Aufgabe 9.4 b)
data Map a b = Map [(a,b)]
instance Listable (Map a b) where
type Item (Map a b) = b
toList (Map []) = []
toList (Map ((a,b) : xs)) = b: toList (Map xs)

--Aufgabe 9.4 c)
data Nat = Zero | Succ Nat
instance Listable Nat where
type Item Nat = ()
toList (Zero) = []
toList (Succ a) = () : (toList a)
```