```
import System.IO

{- 1

LET True = \ x y . x y

LET False = \ x y . y

LET EQU = \ a b . a (\z . b) (b (\z. False) True)

EQU False False =
(\ a b . a (\z . b) (b (\z. False) True)) False False =
(\ b . False (\z . b) (b (\z. False) True)) False =
(\ b . (\ x y . y) (\z . b) (b (\z. False) True)) False =
(\ b . (\ y . y) (b (\z. False) True)) False =
(\ b . (b (\z. False) True)) False =
False (\z. False) True =
(\ x y . y) (\z. False) True =
(\ y . y) True =
True

-}


-- 2

data SList a
  = Co a (SList a)
  | Ni
  deriving (Show,Eq)

allfibs = Co 0 (Co 1 (fib 0 1))
  where
    fib x y = Co (x+y) (fib y (x+y))

-- Jen pro demo

takN _ Ni = Ni
takN n (Co v vs)
  | n<1 = Ni
  | True = Co v $ takN (n-1) vs


{- 3

len []     = 0                -- 1
len (_:xs) = 1 + len xs       -- 2

    [] ++ ys = ys             -- 3
(x:xs) ++ ys = x:(xs ++ ys)   -- 4

dokazujeme
  len (xs ++ ys) = len xs + len ys

xs = []

L = len ([] ++ ys) =|3
  = len (ys) =|elim.nepot.zav
  = len ys
P = len [] + len ys =|1
  = 0 + len ys =|0 je neutral. ve scitani celych cisel
  = len ys
L=P

I.P.
  len (as ++ ys) = len as + len ys

xs = (a:as)

L = len ((a:as) ++ ys) =|4
  = len (a:(as ++ ys)) =|2
  = 1 + len (as ++ ys) =|I.P.
  = 1 + (len as + len ys) =|asoc.+
  = 1 + len as + len ys
P = len (a:as) + len ys =|2
  = (1 + len as) + len ys =|asoc.+
  = 1 + len as + len ys
L=P

Q.E.D.

-}


-- 4

resolve :: Eq a => [a] -> [a] -> Int
resolve xs ys
  | isPrefOf xs ys = -1
  | isPrefOf ys xs = 1
  | True = 0


isPrefOf :: Eq a => [a] -> [a] -> Bool
isPrefOf [] _ = True
isPrefOf (x:xs) (y:ys) = x==y && isPrefOf xs ys
isPrefOf _ _ = False
```

```haskell
-- 5

modify :: String -> String -> IO ()
modify fIn fOut = do
  hIn <- openFile fIn ReadMode
  hOut <- openFile fOut WriteMode
  c <- hGetContents hIn
  hPutStr hOut $ unlines $ procl $ lines c
  hClose hOut
  hClose hIn

procl :: [String] -> [String]
procl (x:y:ys) =
  case resolve x y of
    -1 -> procl (y:ys)
    1  -> procl (x:ys)
    0  -> x : procl (y:ys)
procl x = x


-- EOF
```