```haskell
import System.IO

-- 1)

data AL k d
    = Val k d (AL k d)
    | Nil
    deriving (Show,Eq)

test Nil = True
test (Val k _ rest) = noKey k rest && test rest

noKey _ Nil = True
noKey k (Val k' _ rest) = k/=k' && noKey k rest


-- 2)

fdup :: FilePath -> IO ()
fdup file = do
    h <- openFile file ReadMode
    c <- hGetContents h
    putStr $ unlines $ dl $ lines c
    hClose h

dl :: [String] -> [String]
dl (('+':'+':l):ls) = l:l:dl ls
dl (l:ls) = l : dl ls
dl [] = []


-- 3)

-- apostrofy pouze pro akceptaci v ghci
sum' [] = 0                                 -- 1
sum' (x:xs) = x + sum' xs                   -- 2

foldr' f a [] = a                           -- 3
foldr' f a (x:xs) = f x (foldr' f a xs)     -- 4

{-

sum xs = foldr (+) 0 xs

(1) xs = []

L = sum [] =|1
  = 0

P = foldr (+) 0 [] =|3
  = 0

L=P

2) xs = (a:as)

I.H.
sum as = foldr (+) 0 as

L = sum (a:as) =|2
  = a + sum as

P = foldr (+) 0 (a:as) =|4
  = (+) a (foldr (+) 0 as) =|I.H.
  = (+) a (sum as) =|prefix->infix
  = a + (sum as) =|priorita aplikace nejvyssi -> eliminace zavorek
  = a + sum as

L = P

Q.E.D.

-}
```

```
-- 4)
{-

k - pevny bod
E - vyraz
Y - operator pevneho bodu

Y E = k
E k = k
E (Y E) = Y E

LET mul = \ a b . (iszero a ? 0 : (iszero b ? 0 : mf a b 0))
LET mf = Y (\ f a b r . iszero a ? r : f (pred a) b (add r b))

-}

-- Premie

data Term
    = Var String
    | ValI Integer
    | Term String [Term]
    deriving (Show,Eq)


unify (ValI a) (ValI b) =
    if a==b then Just [] else Nothing
unify (Term a as) (Term b bs) =
    if a==b && length as==length bs then comb [] as bs
    else Nothing
    where
        comb res [] [] = Just res
        comb res (a:as) (b:bs) =
            unify a b >>=
            (\s -> comb (res++s) (map (lapp s) as) (map (lapp s) bs))
unify w@(Var a) t =
    if w==t then Just [] else Just [(a,t)]
unify t w@(Var a) = unify w t

lapp ss t = foldl (flip app) t ss

app (v,t) w@(Var v') =
    if v==v' then t else w
app s (Term t ts) = Term t (map (app s) ts)
app _ x = x

-- EOF
```